DYNAMIC MODELING WITH IMPLICIT SURFACES AND POLYGONAL MESHES

BY

XINLAI NI

B.S., Tsinghua University, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2007

Urbana, Illinois

Abstract

In this thesis, a novel adaptive methods for tracking moving surfaces is presented . Our methods are based on a framework for surface propagation, called *face offsetting*, that moves the mesh faces independently, and reconstructs vertex locations using an eigendecomposition of an error metric together with a viscosity adjustment. Our methods *prevent* local self-intersections, which was a significant roadblock in moving surface meshes. This new framework offers a compelling alternative to level set methods for geometric processing of surface meshes because of its convenience, higher efficiency, and volume conservation.

Utilizing the face-offsetting framework, we also develop a new technique for surface smoothing that preserves volume *locally* (instead of globally) via keeping track of a height function. Volume preservation is critical in many applications, but typical preserving methods rely on a *global* perturbation to the surface and hence suffer from undesirable side effects.

The level sets method propagates a surface represented as an isosurface of a volume of scalar values that automatically accomodates the self-intersection and topology changes that can occur during propagation. The level set method propagates the isosurface in its normal direction according to a user-defined speed function evaluated over it, by deriving and integrating a corresponding time derivative of the voxel values.

We apply the power of the level set method to the propagation of an implicit surface represented not as the interpolation of voxel values but more conventionally through the conglomeration of simpler primitive shapes.

The proposed strategy retains topological benefits of the implicit representation of evolving surfaces while avoiding the drawbacks of a fixed resolution voxel array. This propagation of course occurs within the limits of the implicit's parametrization, and our method creates a least-squares optimal fit of the implicit to the shape specified by the geometric flow.

Morse theory reveals the topological structure of a shape based on the critical points of a real function over the shape. This paper solves a relaxed form of Laplace's equation to find a "fair" Morse function with a user-controlled number and configuration of critical points.

To my parents!

Acknowledgements

This thesis would not have been possible without the support of many people. Many thanks to my advisor, John Hart, who read my numerous revisions and helped make some sense of the confusion.

Table of Contents

List of Tables

List of Figures

1 Introduction

3D models are widely used in graphical applications. There are two main categories of representation of the 3D models, i.e., implicit representation and explicit representation. The implicit representation uses an implicit scalar function $F : \mathbb{R}^3 \rightarrow \mathbb{R}$, the surface is then defined as the zero set $F^{-1}(0)$ of such a function. The explicit representation uses a collection of piece-wise linear patches known as the faces, to store the geometric and topological information. Both representations have advantages and disadvantages, and have been the main subjects of a majority of researchers.

Moving surfaces occur in different forms in computer graphics. They arise as dynamic interfaces in simulations such as water and fire, where the so-called "level sets method" [87] can propagate the surface by changing voxel values appropriately in the simulation grid. However, the geometric flow is defined on the 3D grids, which may easily reach the millions of voxels in modern graphical applications. With the observation that implicit surfaces are controlled by a few parameters, we proposed a formulation that projects the geometric motion of the level sets of a real-value function onto its parametric representation. The resulting method provides a set of new capabilities for implicit models, including computation of geometric flows, such as motion by mean curvature and fitting of implicitly represented surfaces to volumetric data. This method can also be used in fine-tuning the Constructive Solid Geometry construction of the implicit surface, where the motion of different CSG component is guided by the parameter flow.

While level sets methods revolutionized the research in dynamic implicit surface, its application in the explicitly represented surfaces, i.e., polygonal meshes, is limited. We exploited some fundamental ideas used by level sets methods, together with other well-established numerical techniques, to deliver an accurate and stable solution to surface propagation without requiring a 3D voxel grid, known as "face offsetting". A fundamental difference between face offsetting and previous direct surface propagation methods is that our method solves an equation of motion face by face, where continuity and smoothness hold, and then reconstructs vertices by constrained minimization and viscosity adjustment, instead of directly moving vertices along approximate normal directions. As a result, our method prevents the development of local self-intersections, overcoming this significant roadblock in explicit surface tracking. Surface fairing is another important problem in computer graphics. One wants to remove

noise while preserving features from surface models such as those obtained from geometric scanners. Methods that diffuse the normal and fit the surface to the new normals, e.g. [73; 98; 100] have distinct advantages over other easier-to-implement methods based on positional diffusion such as Laplacian smoothing and curvature flows [20; 48]. One of these advantages is volume preservation. Some have sought to preserve the volume globally by rescaling the coordinates of the faired mesh [20], but preserving volume in a local sense is more physically meaningful and visually satisfying. Even normal diffusion methods can lose substantial (in particular, $O(1)$) volume over a large number of iterations on a dense model. Our face-offsetting method, when accompanied by a height function to keep track of the volume, achieves strict local volume preservation.

In 3D modeling, models with higher genus (number of torus holes) are often of particular interests. Morse theory reveals the relation between the topological structure of the 3D surface and the number of critical points of the function defined on the surface. It has been a hot topic in the computational topology community. However, due to the numerical perturbations and scanning errors, real-world models can yield tens of thousands of critical points which became a main hurdle for accurate analysis. Despite its novelty and validity, the scalability of this method is $O(N^2)$. The running time grows dramatically when the complexity of the surface mesh grows. We proposed a linear-time, multigrid algorithm that enables the fairing algorithm to run efficiently and scalably. We can mathematically prove the efficiency of such a multi-grid algorithm, and hence guarantee the linear-time complexity.

This thesis consists of three major parts. In chapter 2, face offsetting method is introduced, with its application in volume preserving smoothing. In chapter 3, we will present the procedural level sets method, the main focus is its adaptation to the CSG models, where a decoupling proposition is proved and applied. With the decoupling proposition, one can break down a large linear system to a number of smaller linear system and thus reduces the computation complexity. A muti-stage surface propagation is also proposed to enable a wider range of surface fitting applications. We conclude the chapter with comparisons to similar methods, in particular, nonlinear least square fitting and direct fitting. In chapter 4, morse fairing method is presented. An original multigrid solution of the fairing is proposed, which gives rise to a linear-time solution for complex meshes.

The thesis is based on the three projects done during my graduate study. As the projects are collaborations with various people, it is necessary to state my contribution to each of the projects. For Face Offsetting, I implemented the system for both local-intersection-free surface propagation and volume preserving surface smoothing and proposed the application of volume preserving surface smoothing based on the principle of face offsetting. For Procedural Level Sets, I contributed on the derivation of propagation speed function on CSG models, and implemented the propagation application based on the publicly available

Wickbert particle system. I also proposed the concept of localizable surface and proved the decoupling proposition which eases the computation on CSG surface propagation. With Morse Fairing, I helped design a multigrid fairing algorithm which reduces the computation complexity from $O(n^2)$ to $O(n)$. I also proved the intermediate value propagation theorem, which relaxes the restriction that the fair morse function should be the strict laplacian solution to the surface function. I also implemented the whole system of Morse Fairing.

2 Face Offsetting

2.1 Introduction

Moving surfaces occur in different forms in a wide variety of graphical applications. They arise as dynamic interfaces in volumetric simulations such as water [74] and fire [70], where the so-called "level sets method" [74; 87] can propagate the surface by changing voxel values appropriately in the simulation grid. They also arise in surface modeling tasks such as smoothing [98], fitting [107], morphing [13] and sculpting [5], where a surface mesh must be artificially embedded into a volumetric distance field to support the stable propagation of the surface using the level sets method. This embedding causes the surface to incur discretization artifacts (such as sigularities) and significantly increases the time and space requirements. Some have sought to propagate a surface mesh directly by imposing a motion on its vertices, e.g. [62; 54], but such methods are prone to numerical problems that severely limit the kinds of surfaces the application could handle.

Here, we present a novel numerical framework for tracking moving meshes, called the *face offsetting method*. This method exploits some fundamental ideas used by level sets methods, together with other well-established numerical techniques, to deliver an accurate and stable solution to surface propagation without requiring a 3D voxel grid. A fundamental difference between face offsetting and previous direct surface propagation methods is that our method solves an equation of motion face by face, where continuity and smoothness hold, and then reconstructs vertices by constrained minimization and viscosity adjustment, instead of directly moving vertices along approximate normal directions. As a result, the proposed method "prevents" the development of local self-intersections, overcoming this significant roadblock in explicit surface tracking. A notable side product of our numerical framework is a vertex-based feature detection technique for surface meshes.

Surface fairing is another important problem in computer graphics. One wants to remove noise while preserving features from surface models such as those obtained from geometric scanners. Methods that diffuse the normal and fit the surface to the new normals, e.g. [73; 98; 100] have distinct advantages over other easier-to-implement methods based on positional diffusion such as Laplacian smoothing and curvature flows [20; 48]. One of these advantages is volume preservation. Some have sought to preserve the volume globally by

rescaling the coordinates of the faired mesh [20], but preserving volume in a local sense is more physically meaningful and visually satisfying. Even normal diffusion methods can lose substantial (in particular, $O(1)$) volume over a large number of iterations on a dense model. Our face-offsetting method, when accompanied by a height function to keep track of the volume, achieves strict local volume preservation.

This chapter is organized as follows. Section 2.2 surveys some related work in the areas of general methodologies for moving surfaces as well as specific techniques for surface fairing. Section 2.3 introduces the basic numerical framework for face offsetting. Section 2.4 presents the use of face offsetting in volume-preserving and feature-preserving fairing. Section 2.5 concludes the chapter with a discussion on future directions.

2.2 Previous Work

2.2.1 Moving Curves and Surfaces

Implicit Surface Capturing Methods. The basic idea of level sets method is to represent a surface with the zero set of a higher dimensional volumetric function and update the function to capture the moving surface. The level sets methods deliver powerful tools for surface propagation, especially at the presence of complex topological changes, by circumventing mesh adaptivity. A large body of research has been done on level sets methods. Comprehensive overviews can be found in the books by Sethian [87] and by Osher and Fedkiw [74]. However, it is difficult to accurately represent the volumetric function near singularities, which limits the discretization accuracy of level sets methods. In addition, these methods are expensive in terms of computation and storage requirements, and they are tedious to set up for complex geometries defined by surface meshes due to the need to convert between implicit and explicit representations.

Besides level sets methods, there are also other implicit methods such as the *volume-of-fluid* (*VOF*) methods. A unique advantage of VOF methods is that they are conservative by construction, which is highly desirable in multiphase flow. Computationally, higher-order VOF schemes are comparable with level sets methods in accuracy and cost, and share some similar limitations such as smearing of the interface [81; 83; 84].

Explicit Surface Tracking Methods. Unlike implicit methods, explicit surface tracking is far less understood. A large amount of work utilizes explicitly moving surfaces. For example, Lawrence and Funkhouser painted a speed function onto an adaptive polygonal mesh to grow trees and other 3D shapes interactively [54], and there are a number of moving-mesh PDE solvers [60]. Yet a surprisingly small number of papers address the fundamental issues with this

method.

One fundamental issue is the treatment of local self-intersections, or "swallwtails" [87, Chapter 4]. These self-intersections are sometimes addressed by trimming (delooping) techniques in 2D [61; 76], but generalization of such techniques to 3D are cumbersome and error-prone and had very limited success. In [47], an entropy-satisfying Lagrangian method was developed to deliver a systematic approach in 2D to prevent local self-intersections by preconditioning the curve before propagating it. The geometric aspect of the face-offsetting method is a natural 3D generalization of that in [47]. Global self-intersections, which correspond to topological changes, pose another major challenge in explicit surface tracking. An extensive survey of mesh surgery at topological changes can be found in [102].

The foremost issue in explicit surface tracking is its potential numerical instability [87, Chapter 4], which has led to doubts about the "tractability" of explicit surface tracking. To the best of our knowledge, our proposed method is among the first to deliver a systematic method that addresses the stability issue of explicit surface tracking in 3D.

Hybrid Methods. To address the limitations of the traditional Lagrangian and level sets methods, a number of hybrid or mixed methods have been proposed in the literature. Some front-tracking methods amend explicit methods with an implicit representation for better robustness [3; 40], which unfortunately introduces smearing into the Lagrangian interface. Some extensions of level sets methods have been proposed to improve their efficiency using narrow-band methods and fast marching methods [87], and to improve conservation using particle level sets methods [74], but the fundamental numerics remain the same. In chapter 3, we will introduce a procedural level sets method that combines the level sets equation with a time parametrization to track the interface using only particles on the interface, which is one step closer to efficient and accurate interface tracking.

2.2.2 Surface Fairing

Surface fairing has also been the subject of much prior work. Most surface-mesh based approaches can be roughly grouped into positional diffusion and normal difffusion. In the former category, Desbrun *et al.* [20] used an area-based curvature normal, and employed an implicit method that takes advantage of the unconditional stability of backward Euler time-integration scheme to allow large time steps. Clarenz *et al.* proposed a modified MCF as an anisotropic diffusion of the surface [17]. In the latter group, Taubin introduced a linear anisotropic Laplacian operator for meshes based on a separate processing of the normals [100], which share similarities with the work by Ohtake *et al.* [73]. A drastically different approach was taken by Jones *et al.* [48], who used statistics and local

first-order predictors of the surface to deliver an efficient non-iterative scheme.

Surface fairing has also been previously performed using the level sets approach. The simplest approach is probably the mean curvature flow [87], which has the side effect of shrinking the volume. Some level-sets based methods use a weighted combination of principal curvatures [86]. Tasdizen *et al.* developed a level sets method that filters normals of the surface and then manipulates the level sets function to fit the processed normals using fourth-order partial differential equations [98]. Numerically discretizing high-order PDEs is difficult. These methods are applied on volumetric data and can be applied to surface meshes after converting them into a volume, which may be expensive. These methods in general take significantly more computation time and storage than surface-mesh based methods.

Among the existing methods, *strict* global volume preservation is achieved in [20] by a simple rescaling operation, which may skew the shape and cannot recover changes in the aspect ratio of a model due to local loss of volume. *Approximate* global volume preservation is achieved in [87] by adjusting the speed of the curvature flow by the average curvature over the whole surface, which can lead to undesirable motion of perfectly flat regions. *Local* preservation is more physically meaningful and is typically adopted by physics-based simulations. However, to our best knowledge, it has not been previously realized in surface fairing.

2.3 Face Offsetting

In this section, we introduce the basic face offsetting method, an accurate and stable numerical framework for moving surface meshes. This method propagates the faces, where smoothness and continuity hold, and then reconstructs vertices using a vanishing viscosity solution. We will briefly describe the basic ideas of the method and focus on only triangular surface meshes.

In the following description, we use u, v, or w to denote a vertex of a mesh, use τ or two vertices (such as uv) to denote an edge, and σ or three vertices (such as uvw) to denote a face (triangle). In numerical computations, upper-case bolds denote matrices, lower-case bolds denote vectors, and regular italics denote scalars. Bolds with hat (such as $\hat{\mathbf{n}}$) denote unit vectors.

2.3.1 Offset Intersections

Throughout the whole document, a *face offset* refers to the new position of a face in the mesh, after moving (offsetting) the face for a distance as dictated by a given speed function and time step. For example, in a mean curvature flow, the face offset may be obtained by moving the vertices along the face normal for a distance equal to a negative factor of the mean curvature. In general, these face offsets around a vertex do not intersect exactly. Our primary task now is

to reconstruct the surface by determining new locations of the vertices so that the face offsets can be best approximated.

Formulation. Consider a vertex v of the mesh. After moving the mesh faces, we want to move this vertex to the new point $\mathbf{x}$ that best matches the surrounding face offsets. Let $\mathbf{h}(\mathbf{x})$ be a vector whose entries contain the signed distance from the point $\mathbf{x} \in \mathbb{R}^3$ to the face offset of each incident face of v. We seek the point that minimizes the weighted squared distance to the face offsets, $\mathbf{h}^T\mathbf{R}\mathbf{h}$. We call this optimal point, denoted by $\mathbf{i}(v)$ or $\mathbf{i}$, the *offset intersection* of v. The face-weighting matrix $\mathbf{R}$ is a diagonal matrix with positive entries.

We locally perform a coordinate transformation at vertex v such that the current location of v becomes the origin. The position vector $\mathbf{i}$ is then equal to the displacement from v to the offset intersection. The *offset direction* of v is the unit vector of $\mathbf{i}$, denoted by $\hat{\mathbf{i}}(v)$, or $\hat{\mathbf{i}}$. We refer to the virtual surface obtained by moving the vertices to their corresponding offset intersections as the *offset surface*, as illustrated in Figure 2.1. We now develop techniques to compute and analyze this offset surface.

Let γ_k be the set of planes passing through each face offset, δ_k denote their signed distance from the origin, $\hat{\mathbf{n}}$ denote the unit outward normals of the face offsets. Then

$$h_k(\mathbf{x}) = \mathbf{x}^T\hat{\mathbf{n}} + \delta_k \tag{2.1}$$

gives the signed distance from any point $\mathbf{x} \in \mathbb{R}^3$.

Let $\mathbf{R}$ be a diagonal weight matrix whose kth diagonal element is r_k, and let $\mathbf{h}$ be the vector of signed distance from $\mathbf{x}$ to the k face offsets, then

$$\mathbf{h}^T\mathbf{R}\mathbf{h} = \sum_k r_k h_k^2(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + 2\mathbf{b}_o^T\mathbf{x} + c \tag{2.2}$$

gives the *fundamental quadric* error that measures the weighted deviation of a point $\mathbf{x}$ to the k face offsets [39]. Note in equation (2.2), $\mathbf{A} = \sum_k r_k\hat{\mathbf{n}}_k\hat{\mathbf{n}}_k^T$, $\mathbf{b}_o = \sum_k r_k\delta_k\hat{\mathbf{n}}_k$ and $c = \sum_k r_k\delta_k^2$. The fundamental quadric is minimized in $\mathbb{R}^3$ by the solution of the 3×3 linear system

$$\mathbf{A}\mathbf{x} = -\mathbf{b} \tag{2.3}$$

Typically, $\delta_k = f_k\Delta t$, where f_k is the speed at the kth face incident on v and Δt is a given time step. In general $\mathbf{A}$ is symmetric and positive semi-definite, and its eigenvalues are all real and nonnegative. Let λ_k be the eigenvalues of $\mathbf{A}$, with $\lambda_1 \geq \lambda_2 \geq \lambda_3$, and $\hat{\mathbf{e}}_k$ be their corresponding orthonormal eigenvectors. These eigenvalues and eigenvectors provide the key to the solution of offset intersections.

Solution of Offset Intersection. If we just solve for the new vertex location using the quadric equation 2.2, we get terrible results. Consider a nearly planar

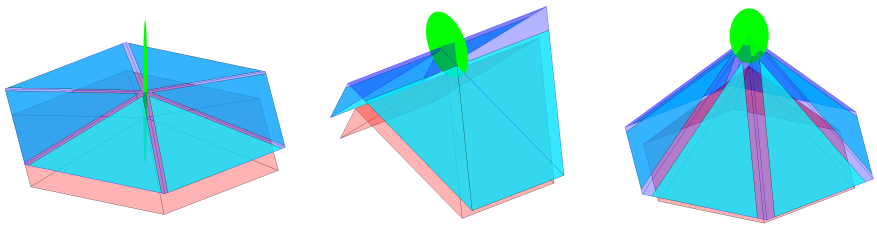

Figure 2.1: Illustration of offset surface around a vertex and the correlation between its local flatness and the relative sizes of eigenvalues. Cyan (upper) triangles depict face offsets of magenta (lower) triangles, which along with light blue patches between them constitute offset surface. Green ellipsoids in the figure are centered at offset intersections, and their axes are aligned along the eigenvectors, with lengths propotional to their corresponding eigenvalues. Note that the error ellipse (error function isosurface) has the same axes but with lengths inversely proportional to the eigenvalues.

region of the mesh being swept forward. Around a vertex on the plane, the face offsets are all nearly coplanar, so the offset intersection can be pushed far across the plane by only a small perturbation of the faces. Hence with a naïve implementation of face offsetting, the mesh will become tangled in planar regions – which is actually where we have the most freedom to move mesh vertices around!

To robustly solve for offset intersections, we must treat smooth, ridge and corner vertices differently. Figure 2.1 suggests a strong correlation between the relative sizes of eigenvalues and local flatness at a vertex. Some authors have previously used eigenvalues alone to detect ridges (creases) in a surface [75], but such a scheme cannot distinguish a very sharp ridge (a near cusp) from a nearly flat surface. We now present a more complete approach.

To classify a vertex v, consider the quadric obtained by setting $\delta_k = -1$ so that the right-hand side of equation 2.3 becomes $\mathbf{b}_m = -\sum_k r_k \hat{\mathbf{n}}_k$, which implicitly uses a coordinate frame whose origin has approximately (in the least square sense) the same distance to each face offset incident on v. We refer to the vector space spanned by the eigenvectors corresponding to relatively large eigenvalues of $\mathbf{A}$ (say $\geq \epsilon\lambda_1$, where ϵ isgarland97a small number such as 10^{-4}) as its primary space – this is the space along which the error function varies most. Let d_m be the dimension of the primary space. we then solve equation 2.3 with right-hand side $\mathbf{b}_m$, i.e., $\mathbf{A}\mathbf{x} = -\mathbf{b}_m$, by forcing $\mathbf{x}$ to lie within the primary space of $\mathbf{A}$ and then obtain $\mathbf{x} = \sum_{k=1}^{d_m} -\hat{\mathbf{e}}_k^T \mathbf{b}_m \hat{\mathbf{e}}_k / \lambda_k$. Its unit vector $\hat{\mathbf{x}}$ is the *mean normal* at v in the offset surface.

Let r_v be the sum of the weights over all incident faces of v. The average angle between a face normal and the mean normal in the offset surface is then

$$\theta = \arccos\left(-\mathbf{b}_m^T \hat{\mathbf{x}} / r_v\right) \tag{2.4}$$

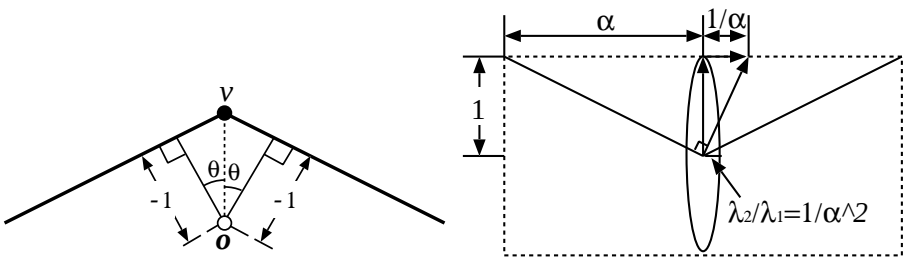

Figure 2.2: Illustration of rationale for feature detection. Left: Mean normal forms approximately same angle with faces. Right: Ratio of eigenvalues is α^2 for quadric associated with minimizing squared distance to diagonals of rectangle with aspect ratio α.

and the average dihedral angle at v is then approximately 2θ, as illustrated in Figure 2.2 (left). Given a threshold $\psi > 0$ for dihedral angles (e.g., 35°) and a singularity tolerance $\mu \in (0, 1)$. As a rule of thumb, $\mu \approx \tan^2(\psi/2)$ (such as 0.1 when $\psi = 35°$) following a rectangle-diagonal argument outlined in Figure 2.2 (right).

After propagation, each vertex v is classified as follows:

- v is smooth if $2\theta \leq \psi$ and $\lambda_3/\lambda_1 \leq \mu$
- v is on a ridge if $2\theta > \psi$ and $\lambda_3/\lambda_1 \leq \mu$
- v is at a corner if $\lambda_3/\lambda_1 > \mu$

Let d_o be the codimension of the tangent space at $\hat{\mathbf{i}}$. In general, d_o is no greater than d_m and is 1, 2 and 3 for a smooth, ridge, and corner point, respectively. The last $3 - d_o$ eigenvectors constitute the *tangent space* of the offset surface at v. After classifying v, the offset intersection $\hat{\mathbf{i}}$ is then computed as $\sum_{k=1}^{d_o} -\hat{\mathbf{e}}_k^T \mathbf{b}_o \hat{\mathbf{e}}_k / \lambda_k$, and the offset direction $\hat{\mathbf{i}}$ is then $\mathbf{i}/\|\mathbf{i}\|$.

2.3.2 Stability Constraint

For PDEs, the time step is restricted by the CFL (Courant-Friedrichs-Lewy) condition, which requires that the computational domain of dependence must contain the physical domain of dependence [45]. For the face offsetting method, this stability condition is satisfied only if the offsets of disjoint faces do not intersect (i.e., free of self-intersection) during a given time interval. There are two types of self-intersections: *local* and *global*. Local self-intersections are accompanied by inversion of triangles, whereas global ones are not. In general, enforcing the CFL condition at the presence of global self-intersections may require detecting the collision of face offsets and then updating the connectivity of the surface. Assuming there are no global self-intersections, we require the preimage of the offset intersection of each vertex v in each of its incident face to

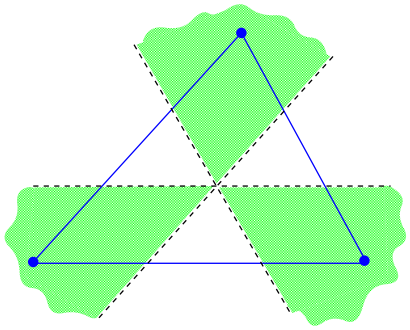

Figure 2.3: Safe regions (shaded areas) for preimages of offset vertices in the original triangle.

be within a *safe region*, where the safe region is bounded by the lines that are parallel to the incident edges of v and pass through the centroid of the face.

Figure 2.3 illustrates the safe regions of the three vertices of a triangle.

Given a face $\sigma = uvw$, let $\mathbf{u}, \mathbf{v}$ and $\mathbf{w}$ be the physical coordinates of its three vertices, $\mathbf{d}$ be a projection direction, and $\mathbf{i}_v$ be the offset i ntersection of v. The preimage of $\mathbf{i}_v$ in σ is then $\mathbf{p} = \mathbf{i}_v - (\mathbf{i}_v^T\mathbf{d})\mathbf{d}$. Let ξ and η be the barycentric coordinates of $\mathbf{p}$ correspondign to the vertices u and w in σ, i.e.,

$$\mathbf{p} = \xi(\mathbf{u} - \mathbf{v}) - \eta(\mathbf{w} - \mathbf{v}) \tag{2.5}$$

If $\max\{\xi, \eta\} \geq 1/3$, then $\mathbf{p}$ falls outside of the safe region in σ. We compute two pairs of ξ and η for each incident vertex-face pair by taking $\mathbf{d}$ to be the face normal before and after propagation. Let α be the maximum value of computed ξ and η for all incident vertex-face pairs. A safe (i.e., fold-over free) time step is then a fraction (e.g., one half) of $\Delta t/(3\alpha)$.

2.3.3 Vanishing Viscosity Solution

The offset intersection provides an approximate new position for each vertex. This approximation is accurate if the physical motion of the surface preserves singularities; however, it overshoots in a diffusive process that should smear out singularities. In addition, the offset intersections of different vertices are computed independently of each other, and this decoupled computation may lead to growing oscillation under small perturbation. We now present techniques to address the overshoot and decoupling issues.

Classification of Motion. We classify the motion of each face at a vertex by comparing each face offset against the offset direction. Given a vertex v and an incident face σ, let $\mathbf{i}_u, \mathbf{i}_v$ and $\mathbf{i}_w$ be the offset intersections of the vertices u, v and w (in counter-clockwise order) of σ. For consistency, we transform $\mathbf{i}_u$

and $\mathbf{i}_w$ to use the local coordinate frame at v, i.e., $\mathbf{i}_u = \mathbf{i}(u) + \mathbf{u} - \mathbf{v}$ and $\mathbf{i}_w = \mathbf{i}(w) + \mathbf{w} - \mathbf{v}$, where $\mathbf{u}, \mathbf{v}$ and $\mathbf{w}$ represents the physical coordinates of the three vertices, respectively. Let $\mathbf{t}$ be the average tangent of the face offset pointing away from v, computed as the unit vector orthogonal to the line segment $\mathbf{i}_u\mathbf{i}_w$, i.e., $\mathbf{t} = \mathbf{i}_u - \mathbf{i}_(u^T\hat{\mathbf{s}})\hat{\mathbf{s}}$, where $\hat{\mathbf{s}} = (\mathbf{i}_w - \mathbf{i}_u)/\|\mathbf{i}_w - \mathbf{i}_u\|$. We say σ is *contracting* at v if $\mathbf{i}_v^T\mathbf{t} \geq 0$, and is *exbanding* otherwise. This is consistent with the traditional classification for convex or concave interface, but has the advantage of embracing saddle points. In addition, a user-input parameter further classifies expansion into *convective* (convection) and *diffusive* (diffusion), where the former preserves but the latter smears out singularities at expansion.

Overcoming Overshoot After classifying a face σ, we compute the contribution of σ to the displacement along the offset direction, denoted by l_σ. If σ is under contraction or convection at v, the intersection is at $\mathbf{i}_v$, and hence $l_\sigma = \|\mathbf{i}_v\|$ (Figure 2.4, Left). If σ is under diffusion at v, then $l_\sigma = \mathbf{i}_v^T\hat{\mathbf{n}} \approx \text{sign}(\mathbf{i}_v^T\mathbf{n})f\Delta t$ (Figure 2.4, Right), where $\hat{\mathbf{n}}$ is the unit normal vector of the face offset pointing away from v, i.e.,

$$\hat{\mathbf{n}} = \frac{(\mathbf{i}_w - \mathbf{i}_v) \times (\mathbf{i}_u - \mathbf{i}_v)}{\|(\mathbf{i}_w - \mathbf{i}_v) \times (\mathbf{i}_u - \mathbf{i}_v)\|} \tag{2.6}$$

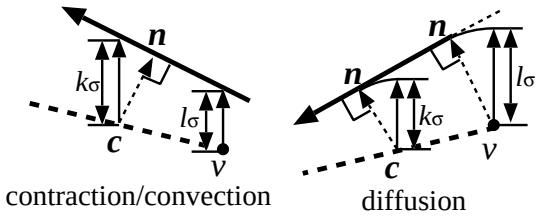

Figure 2.4: Facial contribution and base term for viscosity adjustment along offset direction.

The displacement of v along the offset direction is then computed as a weighted average of facial contributions, i.e.,

$$l = \sum_\sigma \beta_\sigma l_\sigma / \sum_\sigma \beta_\sigma \tag{2.7}$$

We compute the weights β_σ as $\left|\theta\hat{\mathbf{i}}_v^T\hat{\mathbf{n}}\right|$, where

$$\theta = \arccos \frac{(\mathbf{i}_w - \mathbf{i}_v)^T(\mathbf{i}_u - \mathbf{i}_v)}{\|(\mathbf{i}_w - \mathbf{i}_v)\|\|(\mathbf{i}_u - \mathbf{i}_v)\|} \tag{2.8}$$

is the edge angle of the face offset of σ at $\mathbf{i}_v$. If the speed is uniform, or the codimension of v is 1, then $\left|\hat{\mathbf{i}}^T\hat{\mathbf{n}}\right|$ is approximately equal for all incident faces of v and hence has no effect on l. For ridges or corners with a nouniform speed function, this term controls potentially large errors in l_σ when the offset direction is nearly tangent to a face offset (e.g., when the speed function is zero

at one side of a ridge but nonzero at the other side).

Viscosity Adjustment. The preceding weighted average solution resolves the overshoot problem under diffusion, and tends to be exact for contraction and convection. However, it can still be sensitive to perturbation, and may suffer from growing oscillations after a large number of iterations. To stabilize the solution, we must introduce a viscosity term into the method.

Given an incident face $\sigma = uvw$ of v, let $\mathbf{c}$ be the coordinates of the centroid of σ under the local coordinate frame centered at v, i.e., $\mathbf{c} = (\mathbf{u} - \mathbf{v} + \mathbf{w} - \mathbf{v})/3$. We define k_σ as $(\mathbf{i} - \mathbf{c})^T\hat{\mathbf{n}}/(\hat{\mathbf{i}}^T\hat{\mathbf{n}})$ if σ is under contraction or convection, or $(\mathbf{i} - \mathbf{c})^T\hat{\mathbf{n}}$ under diffusion (Figure 2.4). In general, $k_\sigma = l_\sigma + O(s)$ where s is the distance bewteen $\mathbf{c}$ and v, and $k_\sigma = l_\sigma$ if the speed function is uniform and $\mathbf{i}$ is at the exact intersection of face offsets for every vertex. Depending on whether σ is contracting or expanding, we compensate or penalize l_σ based on k_σ. In particular, we take the linear combination $\xi l_\sigma + (1 - \xi)k_\sigma$ as the contribution of σ if this value speeds up contraction or slows down expansion at v, where $\xi = \left|\hat{\mathbf{i}}^T\hat{\mathbf{n}}\right|$ so that the viscosity is 0 (i.e., $\xi = 1$) at the boundary between contraction and expansion. For smooth surfaces, $\left|\hat{\mathbf{i}}^T\hat{\mathbf{n}}\right| = 1 + O(s^2)$, and hence this viscosity adjustment is $O(s^3)$. Lower-order viscosity terms canb e defined by choosing a different α such as $\xi = 1 - \sqrt{1 - \left|\hat{\mathbf{i}}^T\hat{\mathbf{n}}\right|}$, which gives an adjustment of $O(s^2)$. To avoid violating the CFL condition after viscosity adjustment, we limit the adjusted vertex motion to be between 0 and c times of $\mathbf{i}$, where $c \geq 1$ and c times the current time step must remain safe.

The viscosity introduced in this process can effectively help stabilize the surface, especially when $\left|\hat{\mathbf{i}}^T\hat{\mathbf{n}}\right|$ is large,m as it synchronizes the propagation of vertices and faces, analogous to the viscosity terms in the level sets methods. However, for surfaces with sharp ridges, it is necessary to synchronize the propagation of edges within ridges. If the ridge edges are known, it is straightforward to revise the above procedure to compensate or penalize vertex motion based on the motion of edge centers.

We now describe a simple procedure to identify ridge edges in the offset surface based on vertex classification. If $\mathbf{i}_v$ is on a ridge, then the eigenvector $\hat{\mathbf{e}}_3$ is approximately tangent to the ridge, and its incident ridge edges are nearly parallel to $\hat{\mathbf{e}}_3$. In addition, the other vertex of either of its incident ridge edges is most likely also a ridge or corner vertex. Therefore, we identify ridge edges as follows. Let $\hat{\mathbf{t}}_\tau$ denote the unit tangent of an edge τ incident on $\mathbf{i}_v$, i.e., $\hat{\mathbf{t}}_\tau = (\mathbf{i}_u - \mathbf{i}_v)/\|\mathbf{i}_u - \mathbf{i}_v\|$. For each ridge vertex, compute the largest (positive) and the smallest (negative) value of $m_\tau\hat{\mathbf{e}}_3^T\hat{\mathbf{t}}_\tau$, where m_τ is the number of incident ridges or corner vertices of τ. An incident edge is on the ridge if it has either extreme values of $m_\tau\hat{\mathbf{e}}_3^T\hat{\mathbf{t}}_\tau$.

2.4 Volume-Preserving Surface Fairing

The method presented in earlier sections delivers a new general framework for moving surface meshes and is applicable to a large number of applications. To demonstrate its use in applications, we now present a simple adaptation of our method for surface fairing to remove rough features from irregularly triangulated data. This problem is nontrivial as sharp features must be preserved during the process. In addition, it is also highly desirable to preserve the volume of the shape for long-term accuracy and stability. It is straightforward to achieve an artificial global volume preservation by a trivial rescaling [20], but preserving volume in a local sense is more physically meaningful, visually satisfying, but decidedly nontrivial for explicit surfaces.

2.4.1 Diffusion of Normals

Given a vertex v, Equation (2.3) provides a numerical formula for evaluating the mean normal at v when we take its incident faces as face offsets and $\mathbf{b_m}$ as the right-hand side. To diffuse the normal field, we compute the face normal of a given triangle σ as a *weighted* average of those of its vertices. Taking a uniform weights leads to *isotropic* diffusion. Thereafter, let the face offset of σ be the plane passing through the face center with this computed normal, as illustrated in Figure 2.5. We then invoke the face offsetting method. This process repeats until the relative change in the face normal is below a given tolerance, or a maximum number of iterations have been reached.

Intuitively, the above procedure performs a Laplacian smoothing on the surface normal while using face offsetting to reconstruct a surface that accurately approximate the faces. The flow used in the procedure is implicitly curvature dependent, but has done so without requiring explicit evaluation of curvatures – analogous to the secant method for nonlinear equations without requiring the derivatives. This procedure is similar to the mean-filter approach of Ohtake *et al.* [73]. The temporal and spatial adaptivity of face-offsetting collectively ensures the stability of the process with reasonable efficiency.

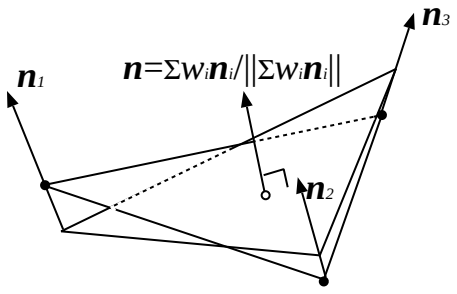

Figure 2.5: Face offset in normal diffusion passes face center with normal equal to average of vertex normals.

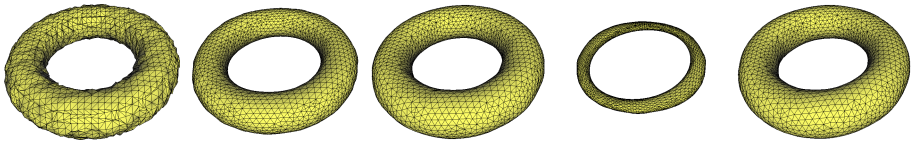

Figure 2.6: Loss of volume is slower asymptotically with normal diffusion than with positional diffusion. Torus with noise (left) were smoothed effectively after 10 iterations for both Laplacian and face-offsetting based fairing (second and third images). Laplacian smoothing leads to nearly 80% volume loss after 100 iterations (fourth) whereas face-offsetting preserved volume (fifth).

2.4.2 Volume Preserving

Besides its simplicity, the normal-diffusion procedure delivers excellent *local* volume conservation. Let s be the longest edge length of a given mesh. For smooth surfaces, the local change in volume associated with coordinate diffusion is $O(s^2)$, where as that associated with normal diffusion is $O(s^3)$. Figure 2.6 shows a comparison in volume changes using positional diffusion and normal diffusion. This higher-order volume preservation suffices for fine meshes, but may still leads to substantial loss of volume for coarse meshes.

To achieve strict conservation, we define a height h for each face and use it to keep track of the volume error at the face. Let $h^{(i)}$ and $A^{(i)}$ denote the height and areas of the face at the ith step, respectively, and $h^{(0)}$ is 0. Then $h^{(i+1)}A^{(i+1)}$ is computed as the sum of $h^{(i)}A^{(i)}$ and the signed swept volume between the old and new positions of the face. CConsider a virtual surface composed of (disjoint) triangles obtained by moving each face along its normal for a distance equal to its height, as illustrated in Figure 2.7. The sum of teh current volume plus the signed swept volume between the virtual and the current surface is invariantly equal to the original volume. To overcome potential instability, we smooth the height function by performing a volume-preserving diffusion: We first transfer the swept volume from each of its incident faces, i.e., $1/3\sum_k h_k A_k$; then, we redistribute the volume from vertices back onto each incident face with a ratio proportional to facial areas.

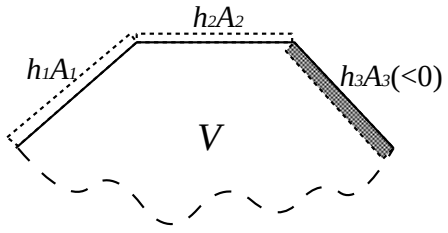

Figure 2.7: Volume preservation through aid of height function. Total volume is V plus integral of height function over surface.

Plugging back into surface fairing, at each step we require the face offset to pass through the face centers of the average of the virtual and actual face (i.e., moving the face center along its facial normal by $h/2$), and then recalculate and diffuse the height function after reconstructing the continuous representation. This procedure delivers strict global conservation in terms of the virtual surface, and the absolute error in the volume of the surface mesh is bounded by the edge length times the total surface area for noise surface but reduces to $O(s^3)$ for smooth surfaces as the process converges. Note that another subtlety in strict volume preserving is edge contraction, where the new vertex must be positioned at a point that loccally preserves the volume. This issue can be ignored when therea are very few or no edge contractions. Figure 2.8 shows the results of smoothing a coarse v-shaped noisy model with and without this height function, where volume was los significantly (more than 10%) after 30 iterations with regular normal diffusion, but is preserved strictly with the virtual surface and consistently (less than 0.25%) with the physical surface. For a model that is twice as fine, the error with normal diffusion was about 3.6% after 100 iterations but is at 0.02% for the physical surface with the height-function extension.

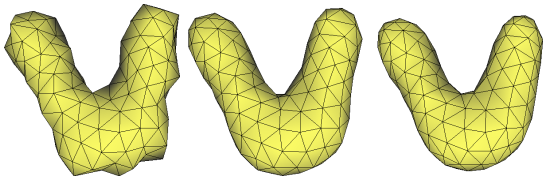

Figure 2.8: After fairing noisy coarse model (left), volume is preserved strictly in local sense using virtual surface extension of face offsetting (middle), whereas more than 10% volume loss occured after 30 iterations with normal diffusion (right).

2.5 Conclusion and Future Work

We have developed a novel framework for moving surface meshes and applied it to volume-preserving and feature-preserving surface fairing. The face offsetting method follows a philosophy analogous to finite volume and discontinuous finite element methods in numerical analysis. It first solves a piecewise continuous solution on each face and then reconstructs a continuous approximation for the surface. Vanishing viscosity was introduced to deliver accurate and stable solutions with high efficiency. This method enables a simple discretization to anormal diffusion approach for surface fairing. We also proposed a new technique to achieve local volume preservation in surface fairing, and this technique can be applied and benefit other graphical applications, such as sculpting.

The method presented here achieves numerical stability by taking a safe time step allowed by the given mesh and speed function. However, the safe

time step may be unreasonably small for poor-shaped meshes and in turn limit the efficiency of the algorithm. Another way to achieve stability is to coarsen the mesh adaptively with a given time step.

Also, the method so far does not yet handle all types of topological changes or topological control. Due to the nature of surface propagation, some kind of topological change could be handled in adaptive face offsetting methods, however, some topological changes (e.g., collision and surgery) are still yet to be considered.

In conclusion, our method is robust in that it prevents local self-intersections, given that the original method is free of self-intersection. Moving surfaces have many applications in computer graphics and computational physica. Another future direction is to iterate our methods to such applications.

3 Procedural Level Sets

3.1 Introduction

The level set method has revolutionized the solution of evolving surface problems in computational science. It propagates a surface represented as an isosurface of a volume of scalar values that automatically accomodates the self-intersection and topology changes that can occur during propagation. The level set method propagates the isosurface in its normal direction according to a user-defined speed function evaluated over it, by deriving and integrating a corresponding time derivative of the voxel values.

With the construction of different speed function, level sets have solved a wide variety of problems in shape modeling and computer graphics. A constant speed function yields offset surfaces [50] and the medial axis transform [51]. Setting the speed function to be the distance to a different object yields a morphing algorithm [14]. More sophisticated speed functions lead to geometry processing algorithm for feature-preserving surface smoothing [99] and scattered point interpolation [71; 26]. Modeling systems have been constructed around level sets where speed functions implement sculpting [6] and blending [69] operators, and the speed function can even be painted on a surface [55].

In all of these examples, the level set method operates on a fixed-resolution uniform rectilinear lattice of voxels. The integration of partial differential equations on a uniform Eulerian space-grid is well studied, and special techniques have been developed to preserve features such as corners [87]. Visual simulations based on computational fluid dynamics often operate on regular space grids, and the level set method has contributed to convincing animations of water [37; 31] and fire [71].

Whereas the voxel space-grid representation is a common choice for CFD-based computer animation, its choice as a shape representation has benefits and drawbacks. A voxel grid provides complete free-form control over shape which has been useful in sculpting simulations, and can be stored efficiently using hierarchical methods [38], and propagated efficiently using the narrow-band method [1]. Nevertheless, the voxel grid is resolution-dependent which restricts its smoothness to the continuity of its interpolant and limits its application in many shape modeling applications.

The goal of this chapter is to apply the power of the level set method to the propagation of an implicit surface represented not as the interpolation of voxel

values but more conventionally through the conglomeration of simpler primitive shapes.

These problems could be avoided by the propagation of a surface mesh instead of a space mesh. Section 3.2 describes previous attempts to propagate surface meshes. While the propagation of mesh vertices is straightforward, the management of mesh connectivity in the presence of self-intersection and topology change become quite complicated.

The proposed computational strategy retains topological benefits of the implicit representation of evolving surfaces while avoiding the drawbacks of a fixed resolution voxel array. To this end, Section 3.3 shows how to propagate an implicit surface (e.g., algebraic surfaces, blobs [11]) under an arbitrary speed function. This propagation of course occurs within the limits of the implicit's parametrization, and our method creates a least-squares optimal fit of the implicit to the shape specified by the geometric flow.

Constructive solid geometry allows the user to create complex surface or object by using boolean operators to combine objects. A CSG model is composed of several simpler shapes, called primitives. The propagation of the whole CSG object can be determined by the propagation of the composing primitives. Section 3.5 shows that a propagation of each primitive is independent of the existence of other primitives, thus we can solve the propagation of the whole object by solving the primitive's propagation one by one.

3.2 Previous Work

The limitations of the voxel representation are well known and some techniques for overcoming time and space complexity, and its fixed-resolution have been investigated.

[87] describes two techniques for reducing the $O(n^3)$ time and space complexity of processing voxel grids. The narrow band method restricts computation to voxels near the isosurfaces, reducing computational complexity to $O(n^2)$. When the speed function does not change sign, the entire propagation can occur in place in amortized $O(n^2)$ time using the fast marching method, which solves for each voxel the time when the surface propagation would pass through it.

A recent GPU implementation [59] accelerates computation and limits storage of the narrow band method via a clever paging mechanism but adds a significant amount of complexity to the implementation.

[87] makes strong arguments against the propagation of surface meshes, presumably through motion of the vertices. When vertices become close, numerical error can cause instability. The local intersection that occurs when propagating corners inward can cause swallowtails. the inside-out curve segments in 2D can be removed by delooping methods [61; 76], but such methods are cumbersome, error-prone and do not appear to extend to surfaces in 3D.

Semi-Lagrangian methods [96; 97] accelerate the propagation of curves in

2D with higher precision by tracing the path of each point on the evolving curve. These paths, called *characteristics*, are the integral curves of the global speed function. The level set value of each space-grid point at time t are thus found as the interpolated level set value at its position at time $t - \Delta t$. Thus the evolving interface moves its surface points along a Lagrangian path, but the path is evaluated at its endpoints on an Eulerian voxel grid.

The entropy satisfying Lagrangian method [47] is able to propagate a closed polyline in 2D. It uses simplification and subdivision to maintain a uniform distribution of curve vertices, and repositioned these vertices at corners to preserve shocks. The method uses a geometric theorem to predict local self-intersections, and locally resamples the curve to avoid the swallowtail before it occurs. The method uses collision detection to detect topology changes, and resolves these with a simple reconnection of the curve vertices. In Chapter 2, we have discussed the extension of this method to 3D.

[55] painted a speed function onto polygonal mesh models to interactively grow tres and other 3D shapes. They propagated a dynamic polygonal mesh [62] but limited their demonstration to simple cases that avoided shocks and topology changes.

The precision of the voxel representation is often limited to a fixed resolution. Recent hierarchical integration methods have been developed that provide a multiresolution space-mesh for evolving a curve in a 2D quadtree [95; 94] or a surface in a 3D octree [79]. In physical simulation and animation, the lack of precision of a space-grid can lead to the inaccurate computation of physical quantities such as volume. When simulating water, the appearance of massive evaporation was avoided by propagating air and water particles near the level set, and adjusting the level set to divide the air and water particles after each propagation step [31; 30].

We demonstrate our results, in part, by using the level set formulation to fit algebraic surfaces to meshed surfaces. Algebraic curve and surface fitting is a well studied area in geometric modeling, and for example Pratt [80] used least squares to directly non-iteratively solve for the algebraic surface coefficients. Our implementation of procedural level sets too is based on least squares, though in our case to minimize the change in surface parameters during surface fitting iterations. Our methods can fit any implicit surface to data, inccluding the blobby model. Procedural level sets provide an iterative alternative to the hierarchical approach of [68].

3.3 Derivation

3.3.1 Problem Statement

Following [87] but in the notation of [104], let $f(\mathbf{x}, \mathbf{q})$ denote a scalar field function over the spatial variable $\mathbf{x} \in \mathbb{R}^3$ controlled by a vector of scalar parameters

$\mathbf{q} \in \mathbb{R}^m$, that implicitly defines the surface

$$S(\mathbf{q}) = \{\mathbf{x} | F(\mathbf{x}, \mathbf{q}) = 0\} \tag{3.1}$$

We extend this formulation to an evolving surface using the shorthand notation

$$S(t) = \{\mathbf{x} | F(\mathbf{x}, t) \equiv F\big(\mathbf{x}, \mathbf{q}(t)\big) = 0\} \tag{3.2}$$

for some path of parameters defined by $\mathbf{q}(t)$.

The level set formulation evolves $S(t)$ along its normals, governed by a real-valued *speed* function $s(\mathbf{x}, t)$ of space, time, curvature, etc.. This surface motion $S(t)$ is implemented by integrating the corresponding change in the field F

$$\dot{F}(\mathbf{x}, t) = -s(\mathbf{x}, t) \| F_{\mathbf{x}}(\mathbf{x}, t) \| \tag{3.3}$$

where $F_{\mathbf{x}}(\mathbf{x}, t)$ is the space gradient of F at $\mathbf{x}$.

The level set method was developed for field functions interpolated from a uniform grid of scalar values f_{ijk}, and the surface evolves by integration of (3.3) evaluated at the grid lattice positions. Our goal instead is to determine the corresponding change $\dot{\mathbf{q}}$ in the field parameters necessary to evolve the surface at the rate specified by the speed function. In our notation, these voxel values f_{ijk} are collected in the $\mathbf{q}$ parameter vector such that (3.3) directly yields the desired change in the surface parameters $\mathbf{q}$, but this is not true for arbitrary field parameter formulations. In order to understand how parameter speed correspond to surface speeds, we will re-derive both the level set equation (3.3) and the implicit surface modeling equations of [104], and show the answer in their combination. Also more generally, we specify the external force or velocity field as a time-varying vector field $\mathbf{v}(\mathbf{x}, t)$ defined at every point $\mathbf{x} \in \mathbb{R}^3$, instead of using a scalar field $s(\mathbf{x}, t)$ and assuming the external force (velocity field) is always along the surface normal direction.

3.3.2 Level Set Derivation

Let $\mathbf{x}^i(t)$ be the path of some particle i on the evolving surface $S(t)$. Then

$$F^i(t) \equiv F(\mathbf{x}^i(t), t) = 0 \tag{3.4}$$

for all t in the surface evolution time interval. We can ensure the path of an initial point $\mathbf{x}(t_0) \in S(t_0)$ remains on the evolving surface by ensuring the time derivative of (3.4) is zero:

$$\dot{F}^i(t) = F^i_{\mathbf{x}}(t) \cdot \dot{\mathbf{x}}^i(t) + \dot{F}(\mathbf{x}^i(t), t) = 0 \tag{3.5}$$

which yield an expression for the change in the function value at a fixed position

$$\dot{F}\big(\mathbf{x}^i(t), t\big) = -F^i_{\mathbf{x}}(t) \cdot \dot{\mathbf{x}}^i(t) \tag{3.6}$$

in terms of the component of the particle's velocity in the surface normal direction. The particle is otherwise free to roam the surface tangentially as it evolves, so when we restrict the particle to move only in the direction of maximum change to F and control its rate with the speed function $s^i(t)$

$$\dot{\mathbf{x}}^i(t) = s^i(t) \frac{F^i_{\mathbf{x}}(t)}{\|F^i_{\mathbf{x}}(t)\|} \tag{3.7}$$

Combining (3.6) and (3.7) yields the level set propagation equation (3.3).

3.3.3 Parametrized Derivation

Let $\mathbf{q}(t)$ represent the shape evolution $S(t)$ as a path through the parameter space of F. This parameter-space path for a level set method is the sequence of grid lattice values produced by integrating (3.3). Here we generalize the derivation to find the desired parameter space path $\mathbf{q}(t)$ for other implicit formulations.

A particle moving along the evolving surface described by $\mathbf{q}(t)$ will satisfy

$$F^i(t) = F\big(\mathbf{x}^i(t), \mathbf{q}(t)\big) = 0 \tag{3.8}$$

which is enforced by the dynamic constraint

$$\dot{F}^i(t) = F^i_{\mathbf{x}}(t) \cdot \dot{\mathbf{x}}^i(t) + F^i_{\mathbf{q}}(t) \cdot \dot{\mathbf{q}}(t) \tag{3.9}$$

where the parametric gradient $F^i_{\mathbf{q}}$ indicates the effect of each parameter on the value of F. Given a shape evolution $\mathbf{q}(t)$, one can thus solve for the paths $\mathbf{x}^i$ that keep particles on the surface. But more powerfully given a reasonable number of particle paths $\mathbf{x}^i(t)$ one can find a shape evolution in the form of a parameter path $\mathbf{q}(t)$ such that at each time t the surface passes through the particle positions.

Equation (3.9) yields:

$$F^i_{\mathbf{q}}(t) \cdot \dot{\mathbf{q}}(t) = -F^i_{\mathbf{x}}(t) \cdot \mathbf{v}(\mathbf{x}^i, t) \tag{3.10}$$

where $\mathbf{v}(\mathbf{x}^i, t)$ is the velocity vector at particle i.

This equation couples the change in parameters $\dot{\mathbf{q}}$ with the speed function. However, because both sides of the equation are real, any number of solutions $\dot{\mathbf{q}}$ can satisfy it. If there are n particles on the surface, and the length of the parameter vector $\dot{\mathbf{q}}$ is m, (3.10) is thus a linear system of n equations and m unknowns. Section 3.4 provides some analysis on this linear system.

3.4 Solution and Analysis

3.4.1 Solve for Parameter Speed

Equation (3.10) is a linear system of n equations and m unknowns. However, for common implicit surface parametrizations ranging from algebraics to blobs, the number of parameters is less, often much less, than the number of particles needed to interrogate the surface.

Subset Fitting. We are tempted to limit our solution to a subset of m floater particles, to yield a full rank linear system. However, the role of the floater particles is to detect variation in the surface and the speed function. While reducing the number of floater particles increases the solution efficiency, it reduces the validity of the solution as important features in the surface or small spikes in the speed function can be missed. The redundancy of the additional floater particles provides the user with an additional implementation variable controlling the resolution of feature sensitivity.

Random Selection. An alternative is to choose a different randomized collection of m particles to solve for the m parameters. One runs the risk of an unlucky choice producign a single system, but such cases could simply trigger the re-choosing of a different random collection of m particles.

Lagrange Multiplier Method. For $n > m$, the linear system (3.10) is *over-constrained*. Thus, there is no accurate solution $\dot{\mathbf{q}}$ satisfying (3.10) for all the n particles. Let A be the matrix on the left-hand side of (3.10), whose ith row is given by the vector $F^i_{\mathbf{q}}(t)$, let $\mathbf{b}$ be the right-hand side vector, whose ith component is $-F^i_{\mathbf{x}}(t) \cdot \mathbf{v}(\mathbf{x}^i, t)$. The Lagrange multiplier method provides a solution to such a problem by finding the "optimal" solution that minimizes the squared error residual $\|A\dot{\mathbf{q}} - \mathbf{b}\|^2$.

The Lagrange multiplier method first expresses $\dot{\mathbf{q}}(t)$ as a linear combination of $F^j_{\mathbf{q}}(t)$

$$\dot{\mathbf{q}}(t) = \sum_j \lambda^j F^j_{\mathbf{q}}(t) \tag{3.11}$$

and then solves for the Lagrange multipliers λ^j by

$$\sum_j \left(F^i_{\mathbf{q}}(t) \cdot F^j_{\mathbf{q}}(t)\right)\lambda^j = -F^i_{\mathbf{x}}(t) \cdot \mathbf{v}(\mathbf{x}^i, t) \tag{3.12}$$

Singular Value Decomposition. An equivalent solution to the Lagrange multiplier method is the singular value decomposition. Generally, for linear system $A\mathbf{x} = \mathbf{b}$ where A has more rows than columns (overconstrained), SVD gives an optimal solution in the least square sense that minimizes the squared residual $\|A\mathbf{x} - \mathbf{b}\|^2$.

We apply SVD to the matrix A in (3.10) to obtain a decomposition $A = USV^T$ where U is an $n \times m$ orthogonal matrix, S is an $m \times m$ diagonal matrix with nonascending singular values and V is an $m \times m$ orthogonal matrix.

The least square solution given by the singular value decomposition is then

$$\dot{\mathbf{q}}(t) = VS^{-1}U^T b \tag{3.13}$$

3.4.2 "Controller" Particles

One might consider avoiding the derivations altogether, and instead implement the standard Witkin-Heckbert control-point particle system [104]. Such controller particles are used to evolve an implicit surface by computing the appropriate parametric velocity that causes the surface to best match the velocity of the controller particles. The parametric motion due to control point motion also occurs in the space spanned by $F_{\mathbf{q}}$, and its Lagrange multiplier are computed using (3.12) and propagate the controller particles, moving them in their normal direction proportional to their speed function. The implicit surface, constrained to interpolate the controller particles, would follow. However, as mentioned before, this would either require a large number of controller particles, or placement of controller particles at key positions on the surface. The floater particles are better suited for this task.

3.4.3 Rank Deficiency

We have observed an overflow in the solution of (3.13), caused by the attempt to inverse the diagonal matrix S which contains one or more zero singular values on the diagonal. The existence of zero singular value in (3.10) is not by accident, it is caused by the fact that all particles we used for constructing (3.10) have zero F value. Actually we have the following result:

Proposition 3.4.1. *Let $\mathbf{x} = (x, y, z)$, then the A matrix of any polynomial (algebraic) surface (in variables x, y, z) is rank-deficient hence induces one or more zero singular values.*

Proof. Since the A matrix has n rows and m columns with $n > m$, to say A is rank-deficient is equivalent to say $\text{rank}(A) < m$, i.e., the m column vectors are linear dependent. On the other hand, a polynomial surface has the form $\sum_j q_j f_j(x, y, z) = 0$, where $f_j(x, y, z)$ is the product of powers of x, y and z. The element A_{ij} is given by $A_{ij} = \partial F(\mathbf{x}^i)/\partial q_j = f_j(x^i, y^i, z^i)$. Note $\sum_j q_j f_j(x^i, y^i, z^i) = 0$ for the particle $\mathbf{x}^i$ is on the zeroset of the implicit function. Thus for the column vectors A_{*j}, we have $\sum_j q_j A_{*j} = 0$, which means the column vectors are linearly dependent. □

Obviously, the proposition follows for arbitrary implicit surface with the form $F(x, y, z) = \sum_j q_j f_j(x, y, z)$ where the subfunction f_j does not contain any $\mathbf{q}$ parameter.

Thus for algebraic surfaces, the straightforward SVD solution is guaranteed to overflow, hence produces bad numerical result. We handle this by introducing a threshold ϵ and ignoring any singular value $s_i < \epsilon$ while taking the inverse of the diagonal matrix S. One can also try adding some particles that have non-zero isovalues.

3.4.4 Regularization

An implicit surface satisfies $F = 0$, but there are infinitely many functions satisfying $aF = 0$ for nonzero real a that evaluate to zero on the implicit surface. We have observed that implicit surface propagation has a tendency to inflate the implicit surface parameters. Furthermore, as the surface nears its goal, these parameters tend to oscillate and thrash as they strive to remove small imperfections.

The least-square optimizations by Lagrange multiplier method should prevent parameter inflation, finding the smallest change in implicit surface parameter necessary to propagate the surface. However, as the evolving surface reaches its target, the speed function f approaches zero.

Global vs. Local Representation. Implicit surface that depend on global formulations, such as algebraics, exponential blobs and thin-plate radial basis functions, create a dense matrix in (3.10), because the function value at each surface point i depends on all parameters $\mathbf{q}$. Local formulations, such as soft object [105], metaballs [72] and compactly-supported radial basis functions [67], whose individual primitives do not contribute outside a given radius, result in a sparse matrix in (3.10), and generally faster solution. Furthermore, in Section 3.5, we will see that the constructive solid geometry model tends to give a block matrix A, one block per CSG primitive, which enables us to solve for $\dot{\mathbf{q}}$ primitive by primitive.

3.4.5 Results

We tested our implicit surface propagation derivations on algebraic surfaces.

Figure 3.1 performs surface fitting and, as a consequence of level set iteration, morphing. We fit an degree-4 algebraic surface by using the target's distance function as the propagation guide. The target point set $\{\mathbf{p}_i\}$ is obtained by sampling the V-shaped model at every vertex. The velocity field is defined as $\mathbf{v}(\mathbf{x}, t) = \alpha(\text{argmin}_i \|\mathbf{p_i} - \mathbf{x}\| - \mathbf{x})$, i.e., proportional to the vector from the query point to the closest point in the target point cloud. Particles on the torus in Figure 3.1 begin inside and outside of the V-shape, and eventually evolve to form the shape of the target V-shape.

Zhao *et al.*'s use of level sets to approximate point sets propagated a first order gradient fit using standard level set propagation, then switched at the end to a second order curvature fit, where the time step is $O(h^2)$. If we instead

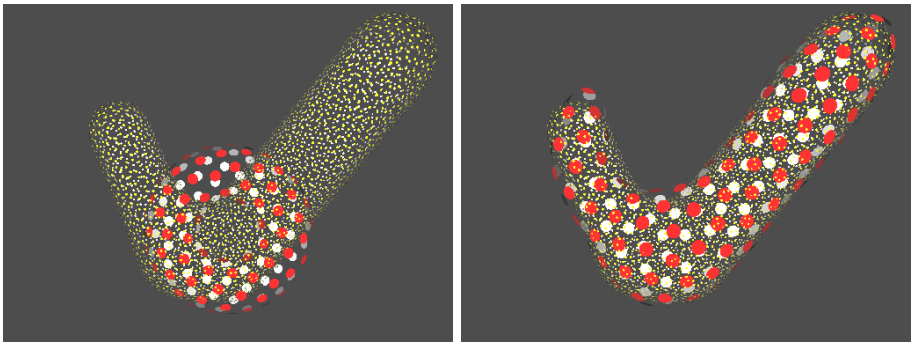

Figure 3.1: A quartic surface propagates to approximate the V-shaped object. Yellow points are the target point cloud sampled from the vertices of the V-shaped triangle mesh. The red particles evolve using the distance vector as the velocity field. The propagation morphs the initial general quartic surface, the torus, into an approximation of the V-shaped object.

use procedural level sets on thin-plate RBF's, then we get minimum curvature automatically.

3.5 Constructive Solid Geometry

Constructive solid geometry allows the user to create complex surface or object by using boolean operators to combine objects. A CSG model is composed of several simpler shapes, called primitives. We will show that the propagation of the combined object can be decomposed into the propagations of the underlying primitives, hence induces an faster solution.

3.5.1 CSG Review

Instead of regarding the surface as the zero set of an implicit function $F(\mathbf{x})$, we can regard it as the boundary subset of a "solid" point set $F(\mathbf{x}) \leq 0$. Thus by taking the pointset operation such as union, intersection or subtraction, one can construct new solid point set from given ones, thus produce new boundary surface.

Define $\text{int}(F) = \{\mathbf{x}|F(\mathbf{x}) < 0\}$ and $\text{ext}(F) = \{\mathbf{x}|F(\mathbf{x}) > 0\}$ to be the *interior* and *exterior* of an implicit function $F : \mathbb{R}^3 \to \mathbb{R}$. Then the union, intersection and subtraction of the interiors of two functions F and G can be computed as:

$$\text{int}(F) \cup \text{int}(G) = \text{int}(\min\{F, G\}) \tag{3.14}$$

$$\text{int}(F) \cap \text{int}(G) = \text{int}(\max\{F, G\}) \tag{3.15}$$

$$\text{int}(F) \setminus \text{int}(G) = \text{int}(\max\{F, -G\}) \tag{3.16}$$

By recursively combining the primitive shapes using the boolean operators above, one can construct fairly complicated shapes. The recursive construction

procedure can be given a binary tree structure, whose leaves are primitives and internal nodes are boolean operations.

3.5.2 Differentiability and R-Functions

Note the boolean operations above are not differentiable at the intersection curve of the surfaces. The procedural level sets method and the Witkin-Heckbert particle system both rely on the differentiability of the implicit function. For approximation, at a point on the surface, we can use the derivative of the corresponding primitive as the derivative of the whole surface. This is well-defined for the point off the intersection curve, but discontinuity occurs at the intersection curve. The theory of R-function was developed by Rvachev in the 1960s [53] for constructing functions that exactly represent virtually any geometric shape of interest in engineering. An R-function is a real-valued function characterized by some property that is completely determined by the corresponding property of its arguments, such as the sign. Shapiro and Tsukanov [88] applied the theory of R-function to construct solid model with guaranteed differential properties.

In particular, for implicit function $F, G : \mathbb{R}^3 \to \mathbb{R}$, we use the R-function

$$R_0^m(F, G) = (F + G \pm \sqrt{F^2 + G^2})(F^2 + G^2)^{m/2} \tag{3.17}$$

This function is m times differentiable even at points with $F(\mathbf{x}) = G(\mathbf{x}) = 0$, i.e., the intersection curve. The plus sign and minus sign correspond to the intersection and union operation of the solid point sets.

Thus using (3.17), we can construct an implicit function, with arbitrary differentiability, whose interior is the union or intersection of the interiors of the given implicit functions F and G.

3.5.3 Propagation of the CSG Surface

We are interested how the CSG surface evolves given a velocity field guidance. Such propagation is of interest because in engineering design, the user knows what kind of primitive shapes to use for construction, but may not know the exact size and relative position. Thus, the user can place the primitives in an approximately correct position with approximate size or orientation, and let the propagation algorithm find the accurate parameters of the composite shape. Since the whole shape is composed with primitive shapes, it makes sense that the propagation of the entire surface is related to the propagation of the individual primitives. Actually, we will show that the propagation of individual primitives are independent with each other, which enables us to decompose the solution to a large linear system to a number of much smaller linear systems.

Localizable Surfaces. Consider a composite implicit function $F(F_1, \cdots, F_k)$, where $F_i \in C^1$ is an implicit function for which we call the ith primitive. Let $\mathbf{q}_i$

be the parameter vector for F_i and let $\mathbf{q}$ be the concatenated parameter vector for F. A point $\mathbf{x}$ is said to be *on the ith primitive* if $F_i(\mathbf{x}) = F(\mathbf{x}) = 0$, but $F_j(\mathbf{x}) \neq 0$ for $j \neq i$. We say such a composite function F is *localizable* if it satisfies the following conditions:

- *Component condition*: $\frac{\partial F}{\partial F_i}|_{\mathbf{x}}$ exists and equals 0 for point $\mathbf{x}$ on primitives other than f_i, and
- *Local infuluence condition*: The derivative of F_i with respect to F_j's parameter $\mathbf{q}_j$ is zero for $j \neq i$, i.e. $F_{i\mathbf{q}_j} = 0$.

Note the CSG operations satisfy the localizable condition. Take the example of the union of two spheres, the compnent condition says if a point $\mathbf{x}$ is on the first sphere, no matter how much it evaluates to the second sphere's implicit function, the entire composite value $F(x)$ will not change (it will always be zero because $\mathbf{x}$ is on one of its primitive component). The local influence condition says the parameters of the second sphere will not affect the first sphere, or the parameters has influence only on their own primitive function.

Constant-Augmented Least Square Solution. We are interested in solving a linear system (3.10) for which we abbreviate as $A\mathbf{x} = \mathbf{b}$. With A having more rows than columns, this becomes an overconstrained system. Instead of finding an accurate solution, we try to find an optimal solution, denoted $A\backslash\mathbf{b}$, minimizing the squared error residual $\|A\mathbf{x} - \mathbf{b}\|^2$. However, if we multiply a nonzero constant c_i on both sides of (3.10), we have a *constant-augmented* version of the linear system $CA\mathbf{x} = C\mathbf{b}$, where C is a diagonal matrix with c_i's on the diagonal. If (3.10) has full rank, the solution to the constant-augmented system is the same as the original system. However, in the overconstrained case, the optimal solution to the constant-augmented version is now $CA\backslash C\mathbf{b}$, called the constant-augmentation of $A\backslash\mathbf{b}$, which minimizes a different squared error residual $\|CA\mathbf{x} - C\mathbf{b}\|^2$. These two residuals are generally different since c_i's can be considered as the weights of different components in the residual vector $A\mathbf{x} - \mathbf{b}$.

The following proposition shows that for a localizable surface, the propagation solution to (3.10) of the whole surface can be decomposed into the propagations of the individual primitives, up to a constant augmentation.

Proposition 3.5.1. *Let $F(F_1, \cdots, F_k)$ be a localizable function. Let $\dot{\mathbf{q}}_i$ be the solution to (3.10) on the ith primitive F_i (using only the floater particles $\mathbf{x}$ on the ith primitive, i.e., $F_i(\mathbf{x}) = F(\mathbf{x}) = 0$ and $F_{j\neq i}(\mathbf{x}) \neq 0$). Then the concatenated vector $[\dot{\mathbf{q}}_1^T, \cdots, \dot{\mathbf{q}}_k^T]^T$ is a constant augmentation of the solution to (3.10) of the entire composite function F.*

Proof. Note although the composite function F may not be differentiable everywhere, it is differentiable at the floater particle $\mathbf{x}$ such that $F_i(\mathbf{x}) = F(\mathbf{x}) = 0$

and $F_{j\neq i}(\mathbf{x}) \neq 0$. Now it is appropriate to apply the chain rule to the composite function F at those points, (3.10) becomes

$$\sum_{i=1}^{k} \frac{\partial F}{\partial F_i} F_{i\mathbf{q}} \cdot \dot{\mathbf{q}} = -\sum_{i=1}^{k} \frac{\partial F}{\partial F_i} F_{i\mathbf{x}} \cdot \mathbf{v}(\mathbf{x}, t) \tag{3.18}$$

Note we have ignored the upper index for the particle and all derivatives in (3.18) are evaluated at the corresponding particle's position.

Consider a particle $\mathbf{x}$ on the ith primitive, due to the component condition of localizable function, the sum in (3.18) is left with only one term, i.e.,

$$\frac{\partial F}{\partial F_i} F_{i\mathbf{q}} \cdot \dot{\mathbf{q}} = -\frac{\partial F}{\partial F_i} F_{i\mathbf{x}} \cdot \mathbf{v}(\mathbf{x}, t) \tag{3.19}$$

By the local influence condition, this further simplifies to

$$\frac{\partial F}{\partial F_i} F_{i\mathbf{q}_i} \cdot \dot{\mathbf{q}}_i = -\frac{\partial F}{\partial F_i} F_{i\mathbf{x}} \cdot \mathbf{v}(\mathbf{x}, t) \tag{3.20}$$

This shows that solving (3.10) for the composite localized function F is equivalent to solving the constant-augmented linear system for each of F's primitive F_i. □

Decoupling the Propagation. Experiments show that a constant-augmented solution does not produce visible deviation from the original least square solution, thus we can solve the propagation of the entire CSG surface by solving the primitives one by one. The only prerequisite condition is that we sample the surface densly enough to guarantee that each primitive has enough particles. In section 3.6, a curvature-dependent sampling scheme is used to guaranee the required sample density. Also, in order for the derivation from (3.18) to (3.19) to be valid, we should avoid using particles straddling on two or more primitives.

3.5.4 Results

We constructed a CSG version of the mechanic part, using the union, intersection and subtraction of 11 leaf primitives. The target point set is obtained by sampling the polygonal mesh. The initial positions and sizes of the composing primitives are deliberately offsetted and biased. Again, we use the vector from the query point to the closest point in the target set as the velocity field. The propagation was solved primitive by primitive. The propagation was shown to converge to the final target point set.

Figure 3.2 shows the sampling of the initial CSG surface. Particles are color coded to indicate which primitive they belong. Figure 3.3 shows the ray-traced renderings of the morphing process.

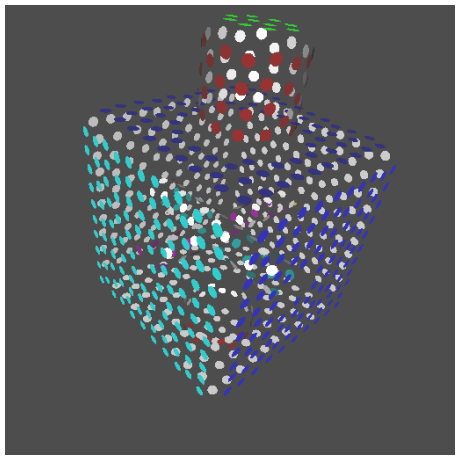

Figure 3.2: The sampling of the initial CSG surface. The mechanic part surface is composed using the union, intersection and subtraction of 11 leaf primitives. The particles are color coded to indicate which primitive they belong.

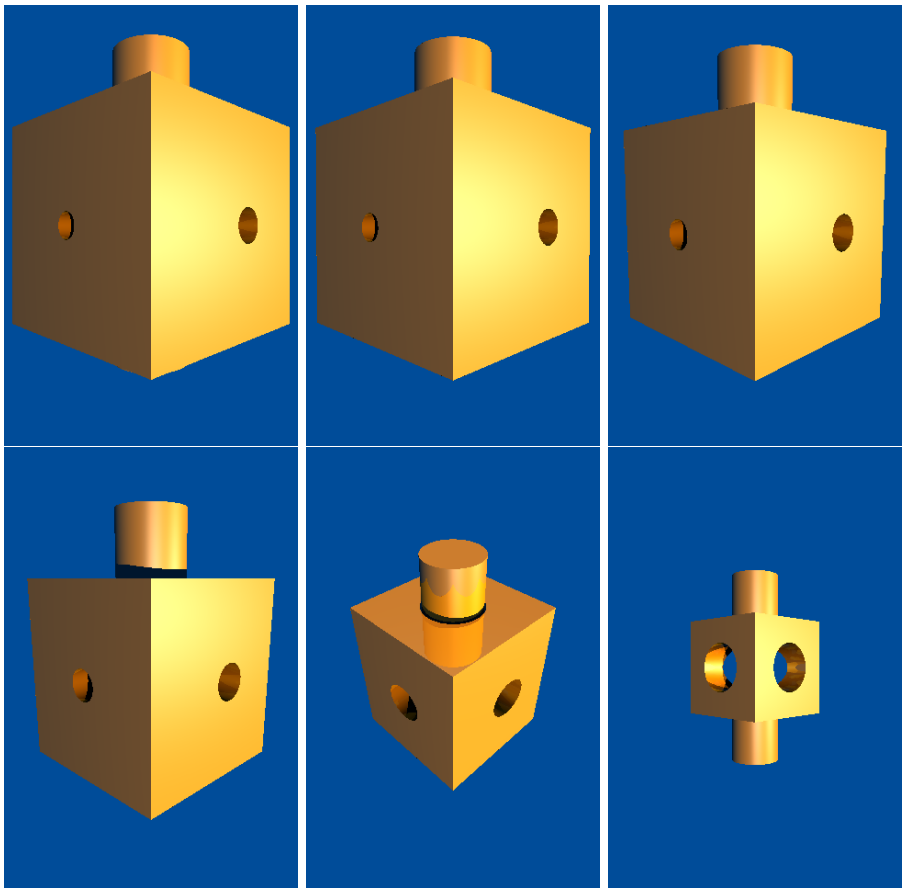

Figure 3.3: The raytraced rendering of the morphing process from a initially offsetted and biased surface to the target point set (not shown).

3.6 Curvature-dependent Particle System

In this section, a curvature-dependent particle system is introduced. The classic Witkin-Heckbert particle system is best for sampling the implicit surface uniformly. However, in the context of surface fitting and procedural level sets propagation, a feature-sensitive, i.e., curvature-dependent sampling of the surface is desired. We propose to modify the WH energy scheme and repulsion-updating criteria to take into account the curvature information of the particles, and achieve a curvature-dependent sampling of the implicit surface.

3.6.1 Review of Witkin-Heckbert System

In [104] Witkin and Heckbert proposed using homogeneous particles to sample and control implicit surfaces. A simple constraint locks a set of particles onto a surface while the particles and the surface moe. The constraint is used to make surface follow particles and to make particles follow surfaces. Control points were introduced for directly manipulating the surface by particle motions. Local repulsion is used to make floaters spread evenly across the surface. Adaptive repulsion radius, together with fissioning and removal of particles are introduced to achieve a good sampling distributions rapidly.

3.6.2 Curvature-Dependent Energy Scheme

In the Witkin-Heckbert particle system, every particle maintains its own repulsion radius σ. The update of σ aims to driving σ to the global desired repulsion radius $\hat{\sigma}$, which is presubscribed. The ideal result of this update procedure yields a hexagonal packing of particles across the implicit surface. The WH particle system uses an energy term that exponentially decreases when the distance $|\mathbf{r}_{ij}|$ between particle i and j increases, i.e., $E_{ij} = \alpha e^{-\frac{|\mathbf{r}_{ij}|^2}{2\sigma_i^2}}$, where E_{ij} is the portion of particle i's energy due to particle j's contribution.

While this yields a uniform distribution of particles across the surface, we may want the distribution to take into account the surface features, most importantly, the curvatures. This is desired since the more curved the surface is, the more information it reveals for the shape of the surface and thus we may want the sampling density to grow at high curvature positions of the surface. Meyer et. al. [63] have proposed an adaptive sampling scheme by changing the energy scheme and using Lavenberg-Marquardt method for fast convergence. However, their method is not suitable in our context because we need to consider the movement of the surface at the mean time, where a gradient descent method is preferred to the LM method. Inspired by their method, we propose an adaptive sampling scheme by a curvature-dependent factor which will affect the mutual repulsion between particles. An analogy can be drawn between the curvature-dependent sampling of particle system and the distribution of charged particles

on a metal shape, where more particles are crowded at tips of the metal conductor. An alternative interpretation of the adaptive sampling density is that, the equilibrium distance between particles should be inversely correlated to the curvature at the particle's position. Consequently, the mutual energy between adjacent particles should be inversely correlated to the curvature.

With this insight, we introduce for each particle i, a curvature factor s_i which is inversely correlated to the curvature κ_i at $\mathbf{x}_i$, and modify the energy scheme as:

$$E_{ij} = \alpha e^{-\frac{|\mathbf{r}_{ij}|^2}{2(s_i\sigma_i)^2}}$$

The effective repulsion radius $s_i\sigma_i$ gives rise to the curvature-dependent equilibrium distance between particles. The curvature factor s_i should be a function of κ_i and decreases when κ_i increases. In the implementation, we let:

$$s_i(\kappa_i) = b + ae^{-c(\kappa_i - \kappa_{min})^2}$$

where the constant a, b and c are determined to satisfy $s_i(\kappa_{min}) = s_{max}, s_i(\kappa_{mid}) = s_{mid}$ and $s_i(\infty) = s_{min}$. We adopt $s_{min} = 0.1, s_{mid} = 0.3, s_{max} = 0.5$ and $\kappa_{min} = 0.5, \kappa_{mid} = 2$.

Since κ_i is a function of $\mathbf{x}_i$, the repulsion force – the partial derivative of E_{ij} with respective to the distance $|\mathbf{r}_{ij}|$, will have a very complex form, usually not analytical. As an approximation, we treat κ_i, hence s_i as constant with respect to $\mathbf{r}_{ij}$, which yields a simpler approximate form of repulsion force:

$$\mathbf{P}_i = \sum_{j \neq i} \mathbf{r}_{ij}\big(E_{ij} + \frac{s_i^2\sigma_i^2}{s_j\sigma_j^2}E_{ji}\big)$$

3.6.3 Curvature-Dependent Adaptive Repulsion

The updating of the particle's repulsion radius is modified to meet the curvature requirement too. We still want that at equilibrium, the arrangement of particles is hexagonal packing, but with one modification that the equilibrium distance, denoted $\hat{\sigma}$, is curvature-dependent. We achieve this goal by mildly decrease the ideal energy $\hat{E}(\kappa)$ for higher curvature κ. Experiment shows that if we let $\hat{E}(\kappa) = 6\alpha e^{-2/\sqrt{s}}$, where $s(\kappa)$ is the curvature factor defined above, the particle system can get to the equilibrium state quickly and correctly. Observe that when $s = 1$, we have $\hat{E} = 6\alpha e^{-2}$ which is the desired energy for a perfect hexagonal packing on the plane.

The modified energy scheme adapts the effective distance between particles by the curvature, and hence introduces an implicit form of scale-invariant energy scheme. Actually since the energy is defined as exponentially decaying function with respect to distance, it could not be scale-invariant, but the introduction of curvature factor ensures that the energy can be approximately treated as scale-invariant within reasonable curvature range. Thus, we no longer need to

presubscribe a desired repulsion radius $\hat{\sigma}$ as in the WH system.

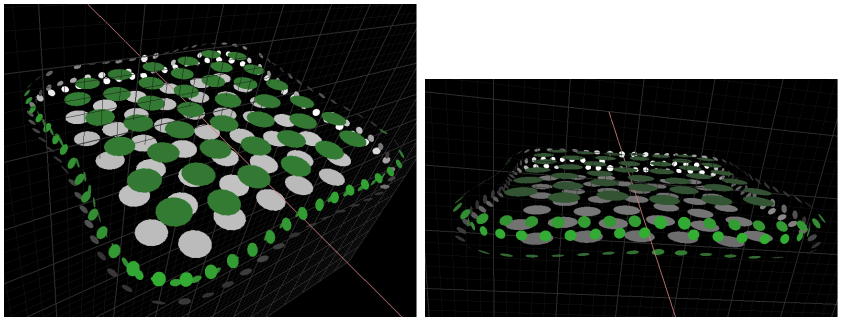

Figure 3.4: A scaled quartic cuboid was sampled adaptively using the curvature-dependent adaptive repulsion.

3.7 Velocity-Field Driven Surface Reconstruction

The problem of surface reconstruction has been of great interests in the community of computer graphics. Scientific data are often represented by scattered individual points, while a closed-form, even algebraic surface is of great importance to further understand the data and provide insights into the relationship bewteen the data points. Also, 3D models are often obtained from 3D scanners, as a set of scattered points, or polygons. Fitting an implicit surface or a set of implicit surfaces to the input data is nontrivial. Algorithms such as MLS are well studied. In this section, we mainly focus on how to fit a set of rather simple-formed implicit surfaces (usually algebraic surfaces of degree no more than 4) to best represent the input point cloud.

3.7.1 Multi-stage Surface Propagation

In the previous sections, we saw the implicit surfaces start from a simple shape, e.g., torus, evolve by the guidance of the attraction of the point cloud, and finally fit the input data points. While this works for evolutions as in the Torus-Tick example, the convergence for arbitrary input and arbitrary starting shapes will fail to yield a satisfactory result. The failure are due to many aspects, for example, the surface may stop evolving because the velocity field generated by the nearby input data point is so strong that the surface is totally ignoring the velocity field generated by far-away points. Thus the one-stage propagation guided by the attraction of input points suffers from local equilibrium. Another reason of the failure of such a velocity field is that due to the complexity of the input points, the expected shape may be convex or concave or irregular. When there are tens of thousands of input points, the velocity field generated by their

attractions is unstable, which means any small perturbation in the initial position or initial velocity of the particles will generate considerably large deviations in the result of the propagation, which has been observed in the experiments. Thus, the so-called "target-attraction" velocity field (or "*kd*-field", as the velocity is calculated from the attractions of the closest points in the input, which is queried using a *kd*-tree) is expected to give the reasonable resulting surface only when:

1. The initial shape with which the propagation start is close enough to the input point cloud.

2. The propagation of the implicit surface is only attracted by the points with which we regard as the target of the surface propagation.

Input Segmentation

While human are capable of identifying the meaningful subsets of the input point cloud, the propagating surface treats them equally. Thus, a primitive targeting the arm of the teddy bear may be attracted to the points near the leg, or even to the outlier points nearby. Therefore, in the first stage, the input point cloud is segmented into meaningful subsets either manually or automatically. In our context where CSG surface fittings are of interests, it is acceptable to manually segment the input point cloud, See Figure 3.5. Here, "meaningful subset" should mean the subset of the input point cloud that can be fairly reconstructed using simple-formed primitive implicit surfaces, such as quadratic or quartic surfaces. The segments are allowed to have overlaps, as some point can be treated as straddling over two or more different meaningful components. After the segmentation is done, each segment generates its own attraction velocity field which later gets used by the propagation of the corresponding primitive surface.

Initial Approximate Fitting

In the second stage, each primitive particle system is associated with a segment from the above segmentation. In addition, the particle system is constrained to an implicit surface which we are going to propagate. The initial parameter vector $\mathbf{q}$ of the implicit surface may be chosen to some predefined values. In our implementation, we always initialize a quadratic surface as a unit sphere, and we provide four different initial $\mathbf{q}$ for quartic surfaces, i.e., torus, cube, cuboid, and a sphere (whose degree-3 and degree-4 terms are trivial). Certainly, this list could be enriched as more interesting featured shapes are recognized and their parameters saved. The more accurately the initial shape approximates the input segment, the better convergence for later propagations. However, it is impossible to list all possible $\mathbf{q}$ as initial choices, but notice that a general ellipsoid can be obtained from a unit sphere by translating, rotating and scaling,

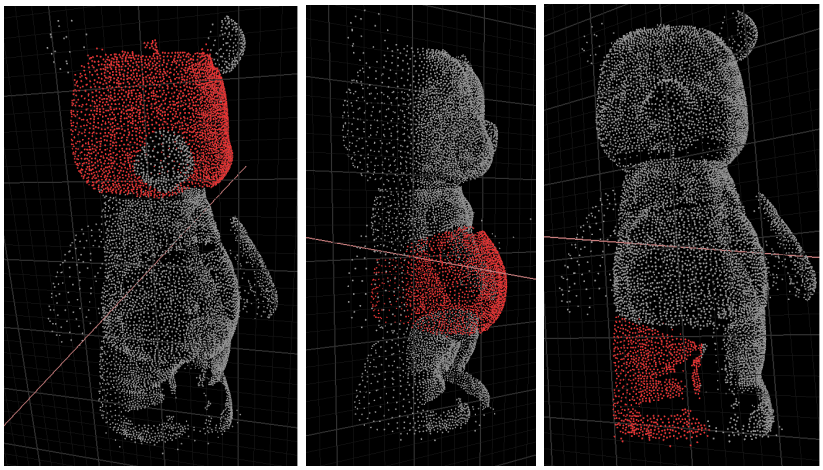

Figure 3.5: Manual segmentation of the scanned teddy-bear model. Selection is implemented fairly easily with OpenGL's selection mode.

so can a cushion-shaped chair seat from a blobby cube. Therefore we allow user to specify the transformation, linear or affine, to be applied to the initial shape so that the shape can approximate the corresponding segment in a even greater accuracy.

Transforming an implicit surface is not trivial, because the transformation is not guaranteed to yield an implicit surface of the same type, even in the case of linear transformation involving only translation, rotation and scaling. For linear transformations, let $\mathbf{M}$ be the transformation matrix, then for implicit surface $F(\mathbf{x}) = 0$, points on the transformed surface should satisfy $F(\mathbf{Mx}) = 0$. It is easy to show that when $\mathbf{M}$ is a linear transformation, and F is an algebraic surface, the transformed surface $F(\mathbf{Mx}) = 0$ has the same degree as $F = 0$. However, when F is not an algebraic surface, e.g., RBF functions, it is hard to write $F(\mathbf{Mx})$ in the form $G(\mathbf{q}, \mathbf{x}) = 0$ whose partial derivative $G_{\mathbf{q}}$ is required by the procedural level sets method.

Figure 3.6 shows the constituent component surfaces before and after the approximate transformation after which a target attraction propagation will lead to a more accurate fitting.

Target Attraction Propagation

In the first two stages, the particle system has been associated to a target group and transformed so that it approximates the target points coarsely. Now we apply the target attraction propagation as described in the previous sections by introducing a velocity field $\mathbf{v}(\mathbf{x}) = \text{argmin}_{|\mathbf{p}_i - \mathbf{x}|} - \mathbf{x}$, where a particle is always attracted to the nearest point in the target group. The query of the nearest point can be done efficiently by constructing a *kd*-tree from the target group. It is easy to see that the particle's desired velocity vanishes only when the particle

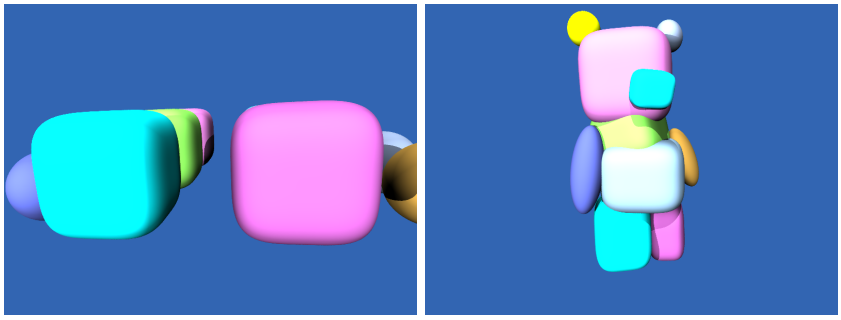

Figure 3.6: Approximate initial transformation. Left: Constituent primitives in their initial positions. Right: Constituent primitives after the initial tranformation.

takes position on one of the target points, but due to surface constraints, one expects the surface to be a best-fit to the target point cloud. See Figure 3.7 for results.

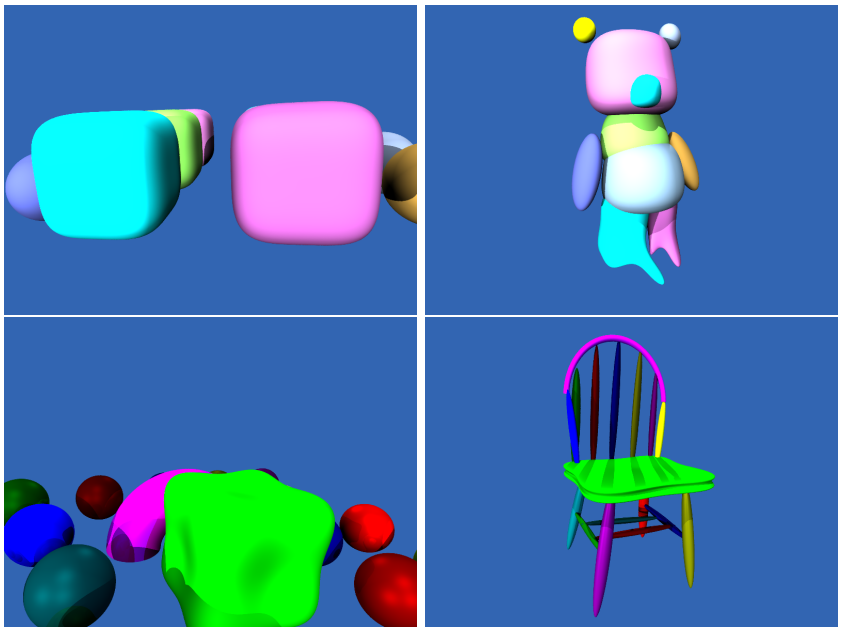

Figure 3.7: Results for surface fitting. Top row shows the surface fitting for the scanned teddy-bear model. Bottom row shows the surface fitting for the chair model. Left: Constituent primitives before fitting. Right: Fitting procedure at equilibrium state.

Local Retouching

The target attraction propagation should have brought the surface to the target point cloud by the end of stage 3. However, the user may still want to locally modify the shape of the resulting surface. A local retouching velocity field is

introduced to meet this need. The retouching field is controled by the "anchor particles", which can be selected by user's mouse click. Let $\mathbf{a}_i$ be the anchor particles for locally pulling the surface outward, and let $\mathbf{b}_i$ be the anchor particles for locally pushing the surface inward. At each anchor $\mathbf{a}_i$ or $\mathbf{b}_i$, a local velocity field is defined as the following:

$$\mathbf{v}_{\mathbf{a}_i}(\mathbf{x}) = \begin{cases} \mathbf{n}_{\mathbf{a}_i}(\mathbf{n}_{\mathbf{a}_i} \cdot \mathbf{n}_{\mathbf{x}}) e^{-\frac{|\mathbf{x}-\mathbf{a}_i|^2}{2\sigma}} & \text{if } \mathbf{n}_{\mathbf{a}_i} \cdot \mathbf{n}_{\mathbf{x}} \geq 0 \\ 0 & \text{if } \mathbf{n}_{\mathbf{a}_i} \cdot \mathbf{n}_{\mathbf{x}} < 0 \end{cases}$$

$$\mathbf{v}_{\mathbf{b}_i}(\mathbf{x}) = \begin{cases} -\mathbf{n}_{\mathbf{b}_i}(\mathbf{n}_{\mathbf{b}_i} \cdot \mathbf{n}_{\mathbf{x}}) e^{-\frac{|\mathbf{x}-\mathbf{b}_i|^2}{2\sigma}} & \text{if } \mathbf{n}_{\mathbf{b}_i} \cdot \mathbf{n}_{\mathbf{x}} \geq 0 \\ 0 & \text{if } \mathbf{n}_{\mathbf{b}_i} \cdot \mathbf{n}_{\mathbf{x}} < 0 \end{cases}$$

where $\mathbf{n}$ denotes the unit outward normal vector at the specified location. Note the velocity contribution of an anchor deminishes exponentially with respect to the squared distance from the anchor, and is forced to vanishes at places whose normal is away from the anchor's normal. The truncation behavior is desirable because a floater particle may be very close to the anchor but is actually on the opposite side of the surface when the surface is very thin. By using the dot-product criterion, we rule out the point who is not local in the geodesic sense.

3.8 Discussion and Comparison

In this section, we compare the procedural level sets method with its two near cousins: *Nonlinear Least Square Fitting* [10] and *Direct Fitting* method [80].

3.8.1 Comparison with Nonlinear Least Square Fitting

The statement of a Least Square Fitting problem is the following:

Given m tagged points whose positions are $\mathbf{x}_i, i = 1, 2, \cdots, m$ and whose tagged values are $y_i, i = 1, 2, \cdots, m$, and given an implicit function $f(\mathbf{x}, \lambda_1, \cdots, \lambda_n)$, we want to determine parameters $\lambda_1, \cdots, \lambda_n$ such that the fitting error

$$\sum_{i=1}^{m} (y_i - f(\mathbf{x}_i))^2$$

is minimized.

The main equation for NLS fitting is:

$$\sum_{j=1}^{n} \frac{\partial f}{\partial \lambda_j}\Big|_{\mathbf{x}_i, \lambda_1, \cdots, \lambda_n} d\lambda_j = y_i - f(\mathbf{x}_i) \tag{3.21}$$

Thus for m points and n parameters, the above yields an equivalent linear system:

$$A_{ij} d\lambda_j = y_i - f(\mathbf{x}_i) \tag{3.22}$$

where the entry of matrix A is $A_{ij} = \frac{\partial f}{\partial \lambda_j}\big|_{\mathbf{x}_i, \lambda_1, \cdots \lambda_n}$.

The statement for Procedural Level Sets is the following:

Given m floater particles whose positions are $\mathbf{x}_i$ with the surface constraints $f(\mathbf{x}_i, \lambda_1, \cdots, \lambda_n) = 0$, how will the surface evolve if the particles $\mathbf{x}_i$ are subject to the velocity influence $\mathbf{v}(\mathbf{x}, t)$?

Recall that our equation is

$$\sum_{j=1}^{n} \frac{\partial f}{\partial \lambda_j}\big|_{\mathbf{x}_i, \lambda_1, \cdots \lambda_n} \dot{\lambda_j} = -f^i_{\mathbf{x}}(t) \cdot \mathbf{v}(\mathbf{x}_i, t) \tag{3.23}$$

where $f^i_{\mathbf{x}}$ is the gradient of particle $\mathbf{x}_i$.

Equation 3.23 can be equivalently written as the linear system:

$$A_{ij}\dot{\lambda}_j = -f^i_{\mathbf{x}}(t) \cdot \mathbf{v}(\mathbf{x}_i, t) \tag{3.24}$$

Now compare 3.22 and 3.24, we can see the major difference between the equations of NLS and PLS is that the right-hand-side vector. In NLS, the right-hand-side denotes the fitting error of each tagged point, while in PLS, the right-hand-side denotes the intention of motion of every particle that's already on the surface.

The idea of these two methods are both to solve the evolution of implicit surface by some guidance. For NLS this guidance is the approximate fitting error $y_i - f(\mathbf{x}_i)$, for PLS, the particle's motion along the normal direction. Since the form of the equations are identical, to some extent, these two methods are equivalent, but as said above, the motivation and guidance are non-trivially different.

3.8.2 Comparison with Direct Fitting

Direct fitting method tries to solve the problem of fitting degree-2 surfaces to the input target points. By inserting the m input points' coordinates into a linear matrix, one obtains $m \times n$ matrix where n is the number of parameters to be determined for the quadratic surface. Using Cholesky decomposition, and deleting the last row of the matrix, one obtains a fully-ranked linear system for which one can solve.

PLS method is suitable for a wider variety of surface fitting than algebraic surfaces. However, in the realm of algebraic surface fitting, their method should be more efficient than ours because Cholesky decomposition is more efficient than the singular value decomposition. On the other hand, in the direct fitting method, the row removed corresponds to the parameter one wished to hold constant, which is hard to determine for complex functions.

3.9 Conclusions and Future Work

We have derived the equations of motion for a general implicit surface evolving under a velocity field sampled on its surface. This enables level set algorithms to avoid a costly voxel representation and use more compact and efficient procedural implicit surfaces instead. A general derivation relating the velocity field to the implicit surface function parameters is stable, but requires a singular value decomposition which can be especially expensive for global implicit surfaces. We also propagate CSG surfaces by decomposing the solution of entire surface to the solution of individual primitives. We have proved such decomposition is differed from the brute-force solution by a constant augmentation, which does not produce visible difference in the propagation process.

We implemented our results usign the publicly available Surface library [44], which implements a generalized, programmable version of surface-constrained particle system [104; 34] and provided an effective development environment for this work.

4 Morse Fairing

4.1 Introduction

Morse theory connects the differential geometry of a surface with its algebraic topology. Given a real function over some shape, it describes the connectedness of the shape from the configuration of the points where the function's gradient vanishes, its so-called critical points (e.g., minima, maxima or saddles). Morse theory has been used in graphics and visualization to analyze different real functions. Terrain data, e.g., is defined by an altitude function on the plane, and Morse theory can identify topographical features, control their simplification [8], and organize them into a multiresolution hierarchy [15]. The zeroset of a real function over 3D space defines an isosurface, and Morse theory can determine its topology for more accurate polygonization [92]. When given only a shape, the critical points of almost all real functions are much better choices than others. The Euler characteristic χ reveals the genus of a closed connected manifold mesh by the formula

$$\chi = \text{vertices} - \text{edges} + \text{faces} = 2 - 2g \tag{4.1}$$

with g the genus of the manifold.

The Euler characteristic can also be calculated from the critical points

$$\chi = \text{minima} - \text{saddles} + \text{maxima} \tag{4.2}$$

By combining these two equations we see that the smallest number of critical points possible on a genus g closed oriented manifold is one minimum, one maximum and $2g$ saddles. However, an arbitrarily chosen Morse function like altitude can yield many more critical points, satisfying the Euler characteristic with any number of additional extrema (minima or maxima) matched by the same number of additional saddles. For example, the altitude function on the genus-6 Buddha model, Figure (4.1) (left) yields 3,605 critical points. A better choice of function (right) yields the minimum number of 14 Morse critical points for this shape. This chapter describes how to find such a function and how to use it to solve various problems in meshed geometry processing.

These extra critical points are caused by the altitude function's sensitivity to surface undulation. A vertical wrinkle in the surface, no matter how small, creates a pair of critical points. These undulations could be removed by smoothing the surface, but it is easier and less destructive to smooth the func-

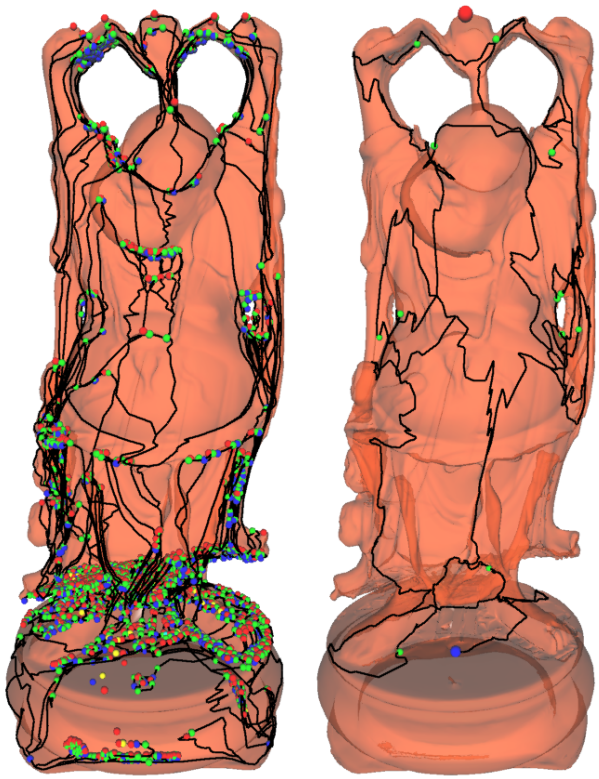

Figure 4.1: An altitude function (left) yields a complicated arrangement of 3,605 critical points on the genus-6 Buddha. Our method yields a fair Morse function (right) with the least number of critical points, in this case one blue minimum, one red maximum and twelve green saddles. Cutting along the indicated path separates the mesh into a shape topologically a disk suitable for planar parametrization.

tion. We leave the surface unchanged, and instead apply Laplacian smoothing to the function to remove its "wrinkles" until it converges to a smooth, in fact harmonic, result that yields the least number of critical points. Because we are removing unwanted function variation over the surface, we call this process *Morse fairing* and the resulting function a *fair Morse* function. Given a fair Morse function, one can trace gradient-descent flowlines down the edges of the mesh from the saddle points to the minimum. These flowlines form a seam that allows the mesh to be cut into a shape topologically equivalent to a disk. Most other methods used in graphics to cut a surface into a disk are based on a region-growing Dijkstra's algorithm. Section 4.2 reveals that Dijkstra's algorithm is in fact an arbitrarily chosen Morse function which leads to extraneous topological events that must be identified and removed. Section 4.3 reviews Morse theory and the state of the art in its application to meshes. This section offers new solutions that make Morse theory work more robustly on meshes, including managing high-multiplicity saddles, the application of Conley index theory to resolve degenerate "flat" regions, and "teflon" saddles that avoid a de-

generate Morse structure. Section 4.4 describes the Morse fairing process, based on solving a constrained Laplacian over the mesh. A proposition in this section leads to a new "intermediate value propagation" multigrid solver that performs Morse fairing in provably linear time, allowing it to be applied in-core to any size mesh, and outruns an irregular-mesh multigrid Laplacian solver. Morse fairing can also produce a real function with auser-specified number of critical points, and can place its extrema in user specified positions. Section 4.3 reviews how the gradient-descent paths of this function embed a graph, called the Morse complex, in the meshed surface that contains a vertex for each minimum, a face containing each maximum and the exact arrangement of edges needed to ensure it matches the topology of the mesh. Section 4.5 describes applications of the faired Morse complex for cutting a surface into a disk, constructing a feature-sensitive topology-correct base domain, clustering faces toward developable charts and visualizing the topology of a complex surface. Section 4.6 concludes with a discussion of the limitations of Morse fairing and directions for further research.

4.2 Previous Work

Cutting a Surface into a Disk. The problem of cutting a closed genus-g mesh of n vertices into a single flattenable component has been investigated as the polygonal schema problem in computational topology [103; 22]. Finding optimal cuts is NP-hard, but cuts with $O(\log^2 g)$ of optimal can be found in $O(g^2 n \log n)$ time [32], and paths through a common base point can be optimized in polynomial time [18]. Arbitrary cuts can be found in $O(gn)$ time [56]. Morse fairing finds the least number of non-optimal cuts through a base point in time linear in the number of vertices, but with an approach that is significantly easier to implement. Mesh topology methods in computer graphics, including topological noise removal [42], geometry images [41], and feature detection [106], find mesh cuts primarily with region growing [24]. In a closed manifold, a front expanding from a base point will self-intersect, and these self-intersections flag the presence of a handle. Front propagation is robust and works well even on meshes with boundaries, and Dijkstra's algorithm can be used when the length of cuts is important. Front propagation generates a real function over the mesh, when the mesh is sufficiently subdivided, the collision of these fronts form Morse critical points [4]. The choice of distance-to-base-points as the Morse function is arbitrary with respect to the surface topology which makes it unnecessarily expensive and prone to generating more critical points than necessary. Morse fairing provides a less expensive real function that generates the same topological information as front propagation, but avoids the maintenance of an priority-queued equidistant front and the expense of collision detection.

Morse Theory on Meshes. Morse fairing was originally devised fro smooth functions on manifolds [64], though it extends elegantly to piecewise linear functions on traingle meshes [9]. Edelsbrunner *et al.* [29] further refined the application of Morse theory to meshes. They first define *persistence* as the difference in value between a pair of critical points that would cancel each other after the appropriate perturbation. Persistence prioritizes the cancellation of critical points, which allows one to control the simplification of features in terrain data and general 2-manifolds [28; 15]. These methods can remove all unnecessary critical points, but do so in order of increasing persistence. Morse fairing leapfrogs this persistence organization and removes unwanted critical points in a single step. The Reeb graph has also been used in graphics to represent shape topology, e.g. [91; 46]. The Reeb graph uses graph topology to represent solid topology (e.g. a Reeb graph cycle represents a torus hole). Similar to us, Steiner & Fischer [93] also used a mesh Laplacian to simplify topology, but instead generated a simplified Reeb graph that lacked extraneous details from non-topological features. The Morse complex represents topology in a surface embeddable structure, and so makes it more amendable to the application of surface processing.

Laplacian Smoothing. Bajaj *et al.* [7] observed that smoothing a Morse function with a Gaussian filter cancelled many pairs of unnecessary critical points, which simplified the critical point structure to aid in the visualization of scientific data. Bremer *et al.* [15] perform iterative Laplacian smoothing steps to cancel critical points in its multiresolution topology hierarchy (and to smooth the jagged 1-cells of the Morse-Smale complex). Ray & Levy [82] devise a multigrid Laplacian solver similar to ours to find a least-square optimal conformal parametrization of a surface triangle mesh. The solution needed for Morse fairing need not be an exact Laplacian, and Section 4.4.3 capitalizes on this property to simplify the multigrid implementation.

4.3 Morse Theory on Meshes

Morse theory relates the homotopy type of a manifold M with its differential structure specified by the critical points of a Morse function $f : M \to \mathbb{R}$. This section reviews Morse theory for smooth functions, then adapts it to the piecewise-linear functions, defining, classfying and using critical points in the absence of derivatives.

4.3.1 Critical Points

Milnor [64] provides a concise, deep but approachable description of Morse theory. Let $\mathbf{p}(\mathbf{u}) \in M \subset \mathbb{R}^3$ be a point on a closed embedded 2-manifold M, in a neighborhood continuously parametrized by $\mathbf{u} = (u_1, u_2)$. Let $f : M \to \mathbb{R}$ be

any real function on the manifold. A point is *critical* if its gradient $[\partial f/\partial u_i]$ vanishes, otherwise it is *regular*. A critical point is *Morse* if its Hessian matrix $[\partial^2 f/\partial u_i \partial u_j]$ is non-singular otherwise it is *degenerate*. If and only if all its critical points are Morse, then the function f is a *Morse function*. Degenerate critical points are unstable; any non-Morse function can be perturbed into a Morse function. The *index* of a Morse critical point is the number of negative eigenvalues of its Hessian V, indicating the number of "downhill" principal directions (eigenvectors). A Morse critical point on a 2-manifold is either an index-0 minimum, an index-1 saddle or an index-2 maximum. Banchoff [9] showed not only that Morse theory extends to triangle meshes, but moreover that its development there is even more elementary than the smooth case. Here f is a piecewise-linear real function. Its values are defined on the vertices of an oriented 2-manifold triangle mesh M, and extend by linear interpolation across the edges and faces of the mesh. For the moment, we assume for each edge $\langle v_1, v_2\rangle \in M$ that $f(v_1) \neq f(v_2)$. Hence the gradient is constant, non-zero and well-defined across the interiors of faces and edges; critical points occur at the vertices. Let $\mathrm{Lk}(v)$ denote the *link* of a vertex v, defined as the graph of m vertices $v_1, v_2, \cdots, v_m$ that share an edge with v, along with the edges $\langle v_1, v_2\rangle, \langle v_2, v_3\rangle, \cdots \langle v_m, v_1\rangle$. We can decompose the link

$$\mathrm{Lk}(v) = \mathrm{Lk}^+(v) \sqcup \mathrm{Lk}^-(v) \sqcup \mathrm{Lk}^\pm(v) \tag{4.3}$$

where $\mathrm{Lk}^+(v)$ is the *upper link* consisting of the vertices $\{v_i \in \mathrm{Lk}(v) | f(v_i) > f_(v)\}$ and the edges $\{\langle v_i, v_j\rangle \in \mathrm{Lk}(v) | f(v_i) > f(v), f(v_j) > f(v)\}$ the *lower link* $\mathrm{Lk}^-(v)$ (replacing $>$ with $<$), and the mixed edges $\mathrm{Lk}^\pm(v) = \{\langle v^+, v^-\rangle | f(v^+) > f(v) > f(v^-)\}$. The number of mixed edges $\#\,\mathrm{Lk}^\pm(v)$, is always even. We hence classify vertices as

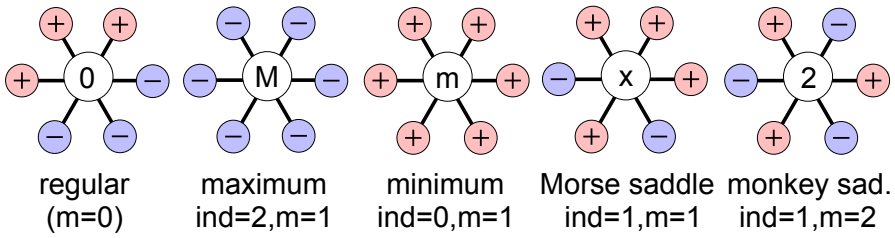

Figure 4.2: Examples of regular and critical vertices.

$$\begin{aligned}
\mathrm{Lk}^-(v) = \emptyset \quad &\Rightarrow \quad v \text{ is a minimum with index 0,} \\
\mathrm{Lk}^+(v) = \emptyset \quad &\Rightarrow \quad v \text{ is a maximum with index 2,} \\
\#\,\mathrm{Lk}^\pm(v) = 2 \quad &\Rightarrow \quad v \text{ is regular,} \\
\#\,\mathrm{Lk}^\pm(v) = 2 + 2m \quad &\Rightarrow \quad v \text{ is a saddle, with index 1} \\
&\qquad\quad \text{and multiplicity } m \geq 1.
\end{aligned}$$

Assigning minima and maxima each a multiplicity $m = 1$ allows us to compute the Euler characteristic as

$$\chi(M) = \sum_{v \in \text{Crit } M} (-m)^{\text{ind } v} \tag{4.4}$$

as proven by Banchoff [9]. A saddle of multiplicity $m = 1$ is a Morse saddle, but in the piecewise linear setting, we have no need for a neighborhood to be locally quadratic, so saddles of any multiplicity (e.g m = 2 monkey and m = 3 dog saddles) can be managed. Edelsbrunner *et al.* [29] demonstrate that the vertex split of a monkey saddle perturbs it into two Morse saddles, but our application can process non-Morse saddles directly.

4.3.2 The Morse Complex

A theorem of smooth Morse theory states that M is homotopic to a cell complex that contains a λ-cell corresponding to each critical points of index λ [64]. If f is Morse-Smale [29] then this 2-complex can be instantiated geometrically as a graph embedded in M. This graph contains a vertex (0-cell) at each minimum, an edge (1-cell) passing through each saddle (with a minimum at each end), and each face (2-cell) contains exactly one maximum. The 1-cells are constructed as integral curves of $-\nabla f$ extending from the two "downhill" sides of the saddle points to the minima. We call this structure the *Morse complex*.

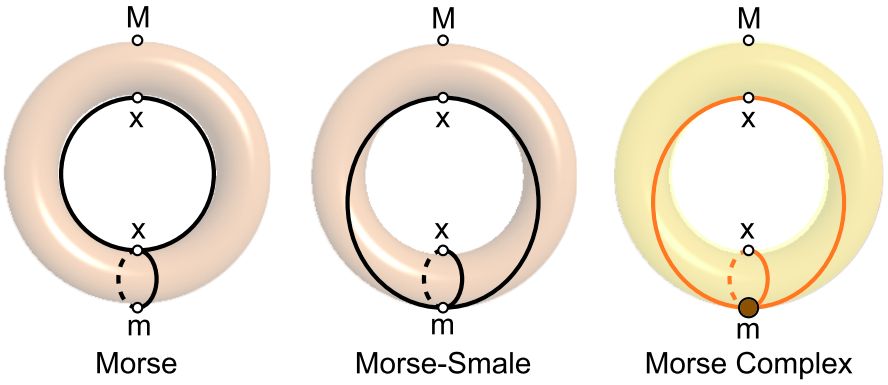

Figure 4.3: Altitude on a vertical torus (left) is Morse, but integral lins flow from the upper saddle to the lower. Leaning the top of the torus forward slightly (middle) yields a Morse-Smale function where integral lines flow from both saddles to the minimum. The Morse complex (right) embeds a brown 0-cell at the minimum, two orange 1-cells along the saddle-point integral lines and a single tan 2-cell in the rest of the torus surface, which contains the maximum.

As before, the embedding of the Morse complex in a smooth manifold also extends to a meshed manifold. For a graph G, let argmin(G) denote the vertex v^- such that $f(v^-) = \min_{v \in G} f(v)$. In the event G has multiple vertices that share the same least value, assume argmin(G) returns only one of them, and

always the same one. The function argmin(Lk(v)) will act as a discrete version of $\nabla(-f(v))$ since it returns the direction of steepest descent. The analog of an integral curve on a mesh is the *flow path*. For a regular vertex v_0, the flow path, flow(v_0) is the chain of vertices $(v_0, v_1, \cdots, v_n)$ and edges $\langle v_i, v_{i+1}\rangle$ such that $v_i = \text{argmin}(\text{Lk}(v_{i-1}))$ and v_n is a minimum. Flow paths do not cross but they can merge, and once merged, they never separate. We do not allow a flow path to penetrate the link of a saddle (other than its origin). If a flow path reaches a vertex v in the link of a saddle v_s, the edge $\langle v, v_s\rangle$ is not considered when computing flow(v). Thus flow paths can approach a saddle but do not reach it, and more importantly cannot cross other paths at the saddle. This procedure consistently and robustly reroutes paths leading into the saddle to paths leading away from the saddle, without requiring the careful ordering of paths prescribed in [29]. These flow paths enable us to embed a 2D Morse

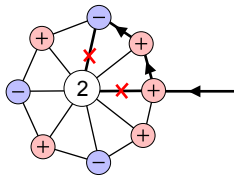

Figure 4.4: Teflon Saddles: Flow paths are not allowed to reach a saddle, and must travel around the saddle's link instead in its quest for a minimum.

complex X into a meshed manifold M. Let X^0 be the set of 0-cells of X, constructed by placing a 0-cell at each minimum in M. Then for each Morse saddle v, decompose $\text{Lk}^-(v)$ into its two disjoint connected components A and B, and let $v_A = \text{argmin}(A)$ and linkewise for v_B. The 1-cell corresponding to saddle v is then the union of flow(v_A), $\langle v_A, v\rangle, \langle v, v_B\rangle$ and flow(v_B) and its ends (minima) are attached to the corresponding 0-cells. The remaining 2-cells, the connected components of $M - X^1$, will each contain a single maximum. We can extend the Morse complex to handle saddles of multiplicity $m > 1$ though it requires the rather inelegant addition of 0-cells at these saddles [1]. First place a 0-cell at each saddle v of multiplicity m. Then decompose $\text{Lk}^-(v)$ into its $m+1$ connected components $\{A_i\}$ and let $v_i = \text{argmin}(A_i)$. For each of these, embed a 1-cell in M corresponding to the flow path flow(v_i), attaching one end to the 0-cell at v and the other to the 0-cell at the flow path's terminating minimum. The two flow paths visiting vertices $(v_1, v_2, \cdots v_k)$ and $(v'_1, v'_2, \cdots, v'_l)$ can merge ($v_i = v'_j$ for some minimal $i < k$ and $j < l$). These merged paths invalidate the embedding, as it becomes two-to-one on their pair of corresponding 1-cells, and many-to-one on the 2-cell between. We can repair this degeneracy of the resulting complex by inserting a 0-cell at the merge vertex v_i and representing the two merged flow paths by a single 1-cell attached to v_i and the terminating

[1] These additional 0-cells do not affect the Euler characteristic because they separate a single 1-cell passing through the saddle into two 1-cells extending from the saddle.

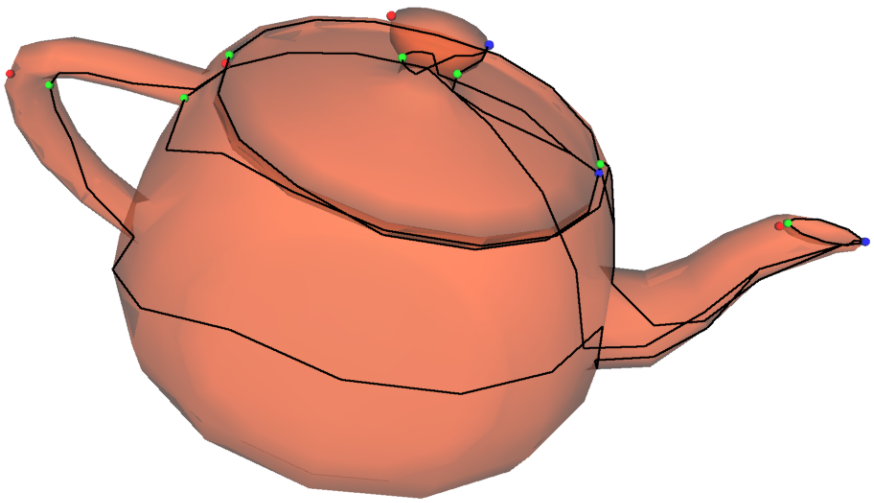

Figure 4.5: The Morse complex of the x-coordinate of a closed-manifold version of the Utal teapot (the handle-to-spout axis). All critical points lie in the xy-plane: blue = minimum, green = saddle and red = maximum.

minimum. Since we have added a 0-cell and a 1-cell, the Euler characteristic remains unchanged, though the resulting cell complex contains additional 0-cells that do not correspond to any critical point.

4.3.3 Flat Regions

The *flat edge* limitation, an edge must have different values on two end points, can be overcome by perturbation, but perturbation can introduce numerous low-persistence additional critical points. A better solution for flat edges can be drawn from Conley index theory [65; 66]. The Conley index simplifies the classification of a complicated critical region in a vector field (more general than the gradient field studied in this chapter) by analyzing the vector field along the boundary of a neighborhood of the critical region, which effectively contracts the critical region into a single point. Let F be a simplified connected maximal collection of equal-valued vertices, as demonstrated in Figure 4.6. Then the boundary of this neighborhood is $\mathrm{Lk}(F)$, where the link is the boundary of the star and in this case is the set of vertices not in F one edge away from a vertex in F, and the cycle of edges connecting them. (Alternatively a flat edge can be processed as a single vertex by an edge collapse, and simply-connected flat region can be converted to a single vertex by a sequence of flat edge collapse.) Thus the lower link can be evaluated and the flat region classified. Since a flat region will have more edges incident on it than would a single vertex, they are more likely to be saddles of high multiplicity, which we can process directly as previously described. The altitude of the Buddha model in Figure 4.1 contains several

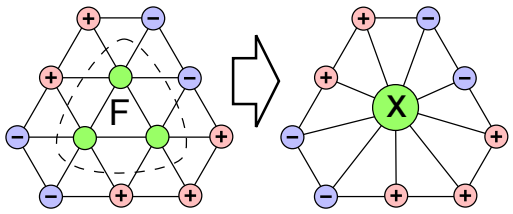

Figure 4.6: A flat region of vertices can be treated as a single vertex. Each of the vertices in the flat region appears to be regular when ignoring flat edges, but contraction of the flat region to a single vertex reveals it to be a saddle.

flat regions that were removed by flat edge collapse. Perturbation of these edges would have resulted in 18 additional critical points. One may encounter a nonsimply-connected flat region, especially in models designed or reconstructed by sampling a rectilinear grid. For example the altitude of the Utal teapot from a triangle mesh derived from its control points contains numerous flat edge cycles. Even the x-coordinate function used in Figure 4.5 contains a single flat edge cycle at its girth. Such cases require more sophisticated methods from Conley index theory that deserve a separate and more in-depth treatment than possible here. Such cases can nevertheless be overcome by perturbation techniques at the risk of additional low-persistence critical points, and the teapot in Figure 4.5 has been rotated slightly. Such nonsimply-connected flat regions are far less common in Morse-faired functions.

4.3.4 Boundaries

We can also extend Morse theory to meshed manifolds with boundary. Such manifolds contain one or more boundaries each represented by a simple closed loop of edges. In preparation for Morse function processing, we sew up each boundary loop by inserting a temporary vertex and creating a face between it and each edge of the boundary loop. The location of the temporary vertex is irrelevant for topological processing, but for geometric concerns can be placed at the centroid of the boundary loop vertices. Likewise the new faces may not generate a valid embedding, but they are only used for combinatorial processing and will be removed once a Morse function has been found. This is similar to the scaffolding triangles of [106] which were used on the much different application of individual patch layout and parametrization. We will assign an extremal function value to each temporary vertex. If the temporary vertex is not constrained to a global minimum or maximum, then one runs the risk of it becoming a saddle. When the temporary vertex and its face neighborhood are removed, the mesh will be missing a key piece of its critical point structure. For cutting the surface into a disk, we make the vertex a minimum, which will connect at least one gradient descent flow to the boundary. When the temporary

vertex and its star are removed, any flow to it will terminate at, and include, the boundary loop. Alternatively, assigning a maximum value to the temporary vertex will encourage (but not guarantee) the flows to avoid the boundary loops, thus isolating and preserving them in the resulting domain. The temporary vertex increments the Euler characteristic by one (the additional edges and faces cancel), as does the additional extremal point. The aggreement of Euler characteristic means that Morse fairing will not introduce any new saddles in response to the new extremum. If edges of a non-manifold mesh are shared by more than two faces, then these edges can be made into a boundary of their incident faces. This however affects the Euler characteristic, leading to an unexpected critical point structure. In the polygon-soup limit where every face is a disjoint component, all topology information is lost and the Euler characteristic reveals no more than the number of faces.

4.4 Finding Fair Morse Functions

Morse fairing is based on the observation that low-pass filtering cancels critical points, which leads to solutions of Laplace's equation that preclude the existence of extrema except at its boundary. This section reviews the Laplacian and its solutions, concluding with a new scalable and very simple multigrid solver that rapidly constructs fair Morse functions given the positions of extrema.

4.4.1 Harmonic Functions and Laplacians

Let f_i denote the value of the real function at the vertex $v_i \in V$. We assume that we are provided with a set $V_C \subset V$ of $k > 0$ constrained points, where f must take on a specified value. Our goal is to construct a suitable function f that has no local extrema other than the constrained points V_C. The lack of local extrema, except at boundary points, is one of the primary properties of emphharmonic functions, which are solutions of the Laplace equation. Thus, our solution to finding a fair Morse function is to find a solution to the Laplace equation $\Delta f = 0$ subject to the Dirichlet boundary conditions imposed by the constrained vertices V_C. The resulting function will be harmonic, and will be guaranteed to be free of extraneous local extrema. The standard definition of the Laplacian operator on a piecewise-linear mesh M is the umbrella operator

$$\Delta f_i = \sum_{\langle i,j\rangle \in M} w_{ij}(f_j - f_i) \tag{4.5}$$

where w_{ij} is a scalar weight assigned to the *directed* edge $\langle i, j\rangle$. It is clear from (4.5) that any vertex for which $\Delta f_i = 0$ has a value f_i which is a weighted combination of the function values at its neighboring vertices. Thus, we can guarantee that f_i is not a local extremum provided that (1) $\sum_j w_{ij} = 1$ and (2) $w_{ij} > 0$ for all edges $\langle i, j\rangle$. These mirror the validity conditions for linear

parametrization methods [36]. We assemble the vertices' function values f_i into an n-vector $\mathbf{f}$ and write the Laplace equation in matrix form

$$L\mathbf{f} = 0 \tag{4.6}$$

where the elements of the $n \times n$ matrix L are given by

$$L_{ij} = \begin{cases} \sum_{\langle i,k \rangle \in M} w_{ik} & \text{if } i = j, \\ -w_{ij} & \text{if edge } \langle i,j \rangle \in M, \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

This matrix L is a *weighted* Laplacian matrix [16]. Note that (4.6) implies $\Delta f_i = 0$ everywhere, though we have subtly flipped a sign convention to ease the remaining derivation. In order to enforce the specified boundary conditions, we construct a constrained (aka "pegged") Laplacian matrix $\hat{L}$ where the row for each constrained vertex is replaced with the corresponding row of the identity matrix:

$$\hat{L}_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } v_i \in V_C, \\ 0 & \text{if } v_i \in V_C, i \neq j, \\ L_{ij} & \text{otherwise.} \end{cases} \tag{4.8}$$

We then solve

$$\hat{L}\mathbf{f} = \mathbf{b} \tag{4.9}$$

where

$$b_i = \begin{cases} f_i & \text{if } v_i \in V_C, \\ 0 & \text{otherwise,} \end{cases} \tag{4.10}$$

which is unique for $|V_C| \geq 2$. Note that this system of equations is entirely equivalent in form to the systems used in linear parametrization methods [36; 19], with the exception that we are solving for a single scalar field. If we constrain exactly two vertices v_i, v_j to field values f_i, f_j such that $f_i < f_j$, then the function f will have a single minimum at v_i, a single maximum at v_j, and by the Euler characteristic, two saddles for each handle in M. Constraining additional vertex values beyond the first two does not always guarantee them to be minima or maxima. For example, constraining three vertices to three different values could yield a solution where the middle-valued vertex is a saddle. To control the number of minima and maxima, it is best to constrain all maxima vertices to the same global maximum value, and all minima vertices to the same global minimum value (so long as no two maxima share an edge, and likewise for the minima).

Selecting Weights. In the linear system (4.9), we have the freedom to choose different schemes for assigning the edge weights w_{ij}. One natural choice are the

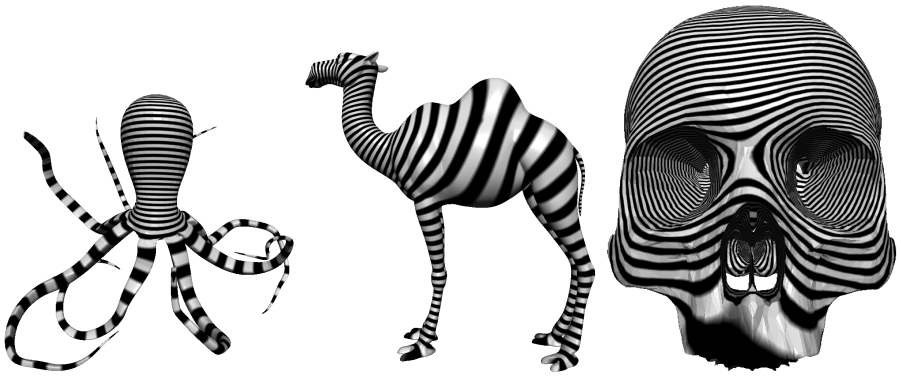

Figure 4.7: Morse functions, both smooth and fair, computed with an irregular multigrid Laplacian solver on a "zebrapus" (max: top, min: tentacle ends), a "cazebramel" (max: nose, min: tail, feet) and a psychedelic skull (max: eyes, min: top & bottom).

combinatorial weights

$$\text{Combinatorial weights:} \quad w_{ij} = 1/\deg(v_i) \tag{4.11}$$

which result in a standard graph Laplacian matrix. This system is purely combinatorial in nature, and these combinatorial weights ignore the geometry of the surface. The graph Laplacian uncovers the topological structure of the surface purely from the connectivity of its mesh. Some applications, such as base domain construction, are sensitive to the geometric properties of the field, and depend on a smooth and well-shaped function over the manifold. These applications are better served by geometry sensitive weight schemes, such as those used in recent parametrization research. The mean value weights

$$\text{Mean Value weights:} \quad w_{ij} = \frac{\tan(\theta/2) + \tan(\phi/2)}{\|v_j - v_i\|}, \tag{4.12}$$

were developed by Floater [35] as a way to approximate harmonic maps while maintaining the convex validity requirement. The values θ and ϕ are the angles the edge $\langle v_i, v_j \rangle$ makes with its two immediate neighboring edges at v_i.

These weights are non-negative which prevents the introduction of unexpected local extrema in the solution. They also produce exceptionally smooth and well-graded scalar fields. Alternative weight schemes that minimize the Dirichlet energy [77; 19] also produce very smooth fields, but can take on negative values in the presence of oblique triangles. This can introduce additional unexpected local extrema in the solution, and makes them unsuitable for our application.

4.4.2 Iterative and Sparse Solutions

Depending on the application and mesh size, a variety of techniques exist to produce a fair Morse function by solving (4.9) or a related system. The simplest of these is the Jacobi iteration

$$\mathbf{f}^{i+1} = D^{-1}\mathbf{b} - D^{-1}W\mathbf{f}^i \tag{4.13}$$

where $\hat{L} = D - W$ decomposes the pegged Laplacian $\hat{L}$ into a diagonal matrix $D_{ii} = \sum_k w_{ik}$ (which inverts elementwise) and the weighted adjacency matrix $W_{ij} = w_{ij}$. For the combinatorial and mean-value weights, $D = D^{-1} = I$, and (4.13) simply replaces each non-pegged vertex with the weighted average of its neighbors while retaining the value of each pegged vertex. In this case one can simply iterate

$$\mathbf{f}^{i+1} = \bar{L}\mathbf{f}^i \tag{4.14}$$

where

$$\bar{L}_{ij} = \begin{cases} 0 & \text{if } i = j \text{ and } v_i \notin V_C, \\ \hat{L}_{ij} & \text{otherwise.} \end{cases} \tag{4.15}$$

is a matrix whose diagonal is zeroed at unpegged rows. Iterating in-place performs a faster Gauss-Seidel solve. However iterated Laplacians converge extremely slowly [43; 52]. For example, mesh noise is often filtered by an iterated unpegged Laplacian on vertex position, e.g. [101; 21], but the limit of this iteration would map all vertices to the same position. Morse fairing is instead interested in the solution of a pegged Laplacian.

Incremental Morse Fairing. When solving a pegged Laplacian, the initial guess f^0 is irrelevant. However, the iteration (4.14) can be used to smooth an initial function. This smoothing reduces and eventually eliminates variation while retaining the characteristics of the original function. This iteration has the effect of cancelling critical point pairs in persistence order [29] though one does not know *a priori* how many iterations are required for cancellation to occur, and cancellations may occur between unexpected critical point pairs. We use iterative Morse fairing later in Section 4.5 to cancel critical points for a curvature-based function.

Sparse Solution. The Laplacian system is sparse, with only $\deg(v_i) + 1$ out of n elements in each row i. This allows sparse solvers, such as SuperLU or netlib's "sparse" package, to be used which can save both space and time for large meshes. Moreover, Morse fairing could be implemented on the GPU by solving (4.9) using conjugant gradients [12].

4.4.3 Multigrid Solutions

For large meshes, even very efficient sparse solution methods can be undesirably slow. Fortunately, the structure of the fields we desire to compute lend themselves very naturally to efficient hierarchical solution techniques. In order to build a suitable mesh hierarchy, we repeatedly coarsen the mesh via edge contraction. In a single pass, we greedily select a maximal independent set of edges to contract. We do this by ranking all edges according to their cost of contraction as determined by the quadric error metric [39]. We then process edges in order of increasing costs, selecting all contractions $v_j \rightarrow v_i$ that meet all of the following validity requirements: (1) the vertex v_j is not a constrained point, (2) Neither v_j nor any of its neighbors are already marked for deletion, and (3) The edge passes the link condition [23] (contracting it will not alter the topology of the surface). The result of this coarsening phase is a sequence of meshes $M^0, M^1, \cdots, M^k$, where M^0 is the original mesh, and M^i is the mesh after the i^{th} coarsening step. The simplest mesh, M^k, we refer to as the base domain mesh.

Having produced a suitable base domain, we can solve a simplified constrained Laplacian system (4.9) on it. Note that by construction, all constrained vertices must still be present in the base domain, and it must have the same genus as the input. However, unless an unusually large number of vertices are constrained, the base domain can have a very simple structure. For example, the minimal base domain for a genus-0 mesh with 2 constrained points is a tetrahedron. Even for the turbine blade model (Figure 4.13) with 295 separate connected components and 165 handles, the minimal base domain has only 2,744 triangles and 1,632 vertices. Any reasonably efficient sparse matrix solver can solve systems of this size in a small fraction of a second on modern hardware.

Irregular Multigrid

Having computed an approximate solution field on the base domain M^k, we want to extend this solution back to the original mesh M^0. We do this by a standard irregular multigrid approach quite similar in form to those recently developed independently by Aksoylu *et al.* [2] and Ray and Levy [82]. We repeatedly refine the mesh, undoing the contractions performed during coarsening. During each refinement phase, we perform vertex splits corresponding to all the contractions performed during the corresponding coarsening phase. Our validity rules guarantee that, when a vertex v_i is reintroduced into the mesh during refinement, the approximate solution field already exists at all of its neighbors. Our initial estimate for the field value f_i at the new vertex v_i is simply formed by the linear combination of its neighbors $f_i = \sum_j w_{ij} f_j$. This produces an approximate solution, which we must iteratively relax (§4.4.2) until convergence. This multigrid algorithm produces the same solution field as a full matrix solver (subject to the convergence tolerance used) as demonstrated

in Figure 4.7. However, its running time is dependent on the quality of the intermediate mesh approximations, and is not guaranteed to be $O(n)$ if the approximations are poor[2].

Intermediate Value Propagation

Recall that no unpegged vertices of the base domain M^k are extremal. The multigrid Laplacian solver performs a relaxation after each vertex split is to ensure that the function value assigned to each re-introduced vertex creates no new extrema. In fact our interest in the Laplacian is for its elimination of extrema. The following theorem shows that we can avoid extrema during refinement without constructing an expensive Laplacian solution, leading to the *intermediate value propagation* algorithm.

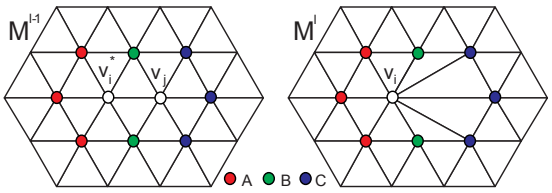

Figure 4.8: Neighborhood of a vertex split.

Proposition 4.4.1. *Let non-extremal vertex v_i split into edge $\langle v_i^*, v_j \rangle$ and assign $f(v_i^*) = f(v_i)$. Define the open intervals*

$$U = (f(v_i), \min_{v \in \mathrm{Lk}^+(v_i)} f(v)), \tag{4.16}$$

$$L = (\max_{v \in \mathrm{Lk}^-(v_i)} f(v), f(v_i)), \tag{4.17}$$

and let

$$A = \mathrm{Lk}(v_i) \setminus \mathrm{Lk}(v_j) \tag{4.18}$$

$$B = \mathrm{Lk}\,\langle v_i^*, v_j \rangle \tag{4.19}$$

$$C = \mathrm{Lk}(v_i) \setminus \mathrm{Lk}(v_i^*) \tag{4.20}$$

categorize the vertices sharing an edge with v_i. Then setting the value of the new

[2]Poor approximations are in general unavoidable as no provably-good surface simplification algorithms are yet known.

vertex v_j such that

$$f(v_j) \in \begin{cases} L & \text{if } \mathrm{Lk}^-(v_i) \subset C, \quad \textit{(Case 1)} \\ U & \text{if } \mathrm{Lk}^+(v_i) \subset C, \quad \textit{(Case 2)} \\ L & \text{if } \mathrm{Lk}^+(v_i) \subset A, \quad \textit{(Case 3)} \\ U & \text{if } \mathrm{Lk}^-(v_i) \subset A, \quad \textit{(Case 4)} \\ U \cup L & \text{otherwise,} \quad \textit{(Case 5)} \end{cases} \tag{4.21}$$

creates no new extrema.

Proof. Vertex v_i is not extremal: Caser 1: $\exists v \in \mathrm{Lk}^+(v_i) \cap (A \cup B)$ such that $f(v_j) < f(v_i) < f(v)$. Case 2: $\exists v \in \mathrm{Lk}^-(v_i) \cap (A \cup B)$ such that $f(v) < f(v_i) < f(v_j)$. Cases 3–5: $\exists v_1 \in \mathrm{Lk}^-(v_i) \cap (A \cup B), v_2 \in \mathrm{Lk}^+(v_i) \cap (A \cup B)$ such that $f(v_1) < f(v_i) < f(v_2)$. Vertex v_j is not extremal: Cases 1, 2 and 5: $\exists v_1 \in \mathrm{Lk}^-(v_i) \cap (B \cup C), v_2 \in \mathrm{Lk}^+(v_i) \cap (B \cup C)$ such that $f(v_1) < f(v_j) < f(v_2)$. Case 3: $\exists v \in \mathrm{Lk}^-(v_i) \cap (B \cup C)$ such that $f(v) < f(v_j) < f(v_i)$. Case 4: $\exists v \in \mathrm{Lk}^+(v_i) \cap (B \cup C)$ such that $f(v_i) < f(v_j) < f(v)$. Vertices in A do not become extremal since their links do not change. Vertices in B do not become extremal since v_j is simply added to their links. Vertices in C do not become extremal, for if v_i was a lesser or a greater neighbor of any vertex in C, then for all five cases setting $f(v_j) \in L \cup U$ will continue that role. □

For meshes with bounded valence, single vertex simplification and refinement are constant-time operations. Therefore, a multigrid fairing algorithm using this refinement operation runs in time linear in the number of vertices, making Morse fairing a scalable procedure up to the size constraints of main memory. This refinement results in a Morse function with the least number of critical points.

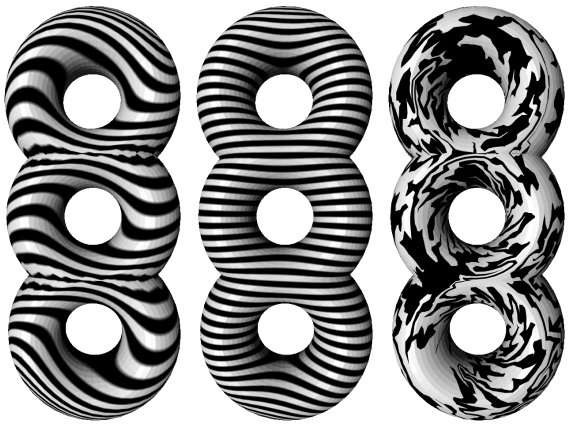

Figure 4.9: Combinatorial weights (left) ignore the geometry of the mesh, and can produce undesirable field variation (though no new critical vertices). Mean value weights (middle) produce a very smooth field. Our new intermediate value propagation solution produces a very random field (right), but surprisingly (and provably) no new critical points.

4.4.4 Discussion

Properly accounting for the geometry of the surface has a substantial effect on our resulting fields, as demonstrated in Figure 4.9. Using the purely combinatorial weights $w_{ij} = 1/\deg(v_i)$ produces a field with a great deal of unwanted variation. In contrast, the mean value weights produce a very smooth, pleasingly symmetric field. Intermediate value propagation picks an arbitrary value within bounds during refinement which leads to wild fields that nevertheless avoid extrema and by the Euler characteristic result in the least necessary number of saddles.

Model	Vertices	Mean-Value Laplacian Multigrid (s)	Intermediate Value Propagation Multigrid (s)
Teapot	553		0.050
V2	1,923	1.36	0.220
3-torus	4,236	1.99	0.471
Camel (S)	11,225	4.8	1.48
Cranium	12,365		1.76
Octopus	16,554	11.9	2.43
Bunny	34,834		6.17
Camel (M)	44,897	25.9	6.85
Horse	48,485		7.51
Turbine	50,260		8.49
Camel (L)	179,585	110.	34.8

Table 4.1: Multigrid solver execution times (256MB 1.2GHz P3).

Table 4.1 compares the running times of the two multigrid approaches. Intermediate value propagation avoids the relaxation step that a multigrid Laplacian solution requires, resulting in at least a threefold speedup. Figure 4.10 shows the scalability of both multigrid solvers.

4.5 Applications

Having derived a more robust Morse theory for meshes and a fast algorithm for fair Morse function construction, we now turn to its application to problems important in computer graphics.

4.5.1 Cutting a Surface into a Disk

Often a closed meshed surface needs to be cut into a single flattenable piece, which can aid surface texturing, deformation, integration, navigation, sampling and storage, and is a key ingredient of, for example, the recently popular geometry images technique [41]. Once cut, the mesh can be flattened into a compact subset of the plane using any number of existing techniques, e.g. [36; 78; 89]. We cut a surface into a disk by assigning a single minimum that will serve as the base point, and a single maximum representing the single face in the Morse complex. These choices can be made arbitrarily, e.g. as the vertices with the lowest

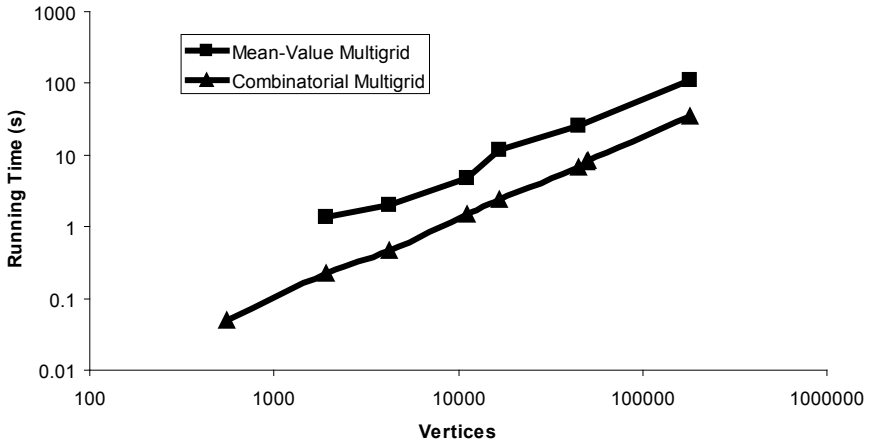

Figure 4.10: Both the mean-value and propagation multigrid solvers scale linearly (slope of both log-log graphs is one) making them appropriate for in-core processing of any size mesh.

and highest altitude. We then solve the constrained Laplacian, which yields a Morse function with two saddle points for each handle. Each saddle will have a cycle (pair of gradient descent paths) extending from it to the minimum. Each handle will generate two non-separating cycles, one around the hole, and one around the handle. Both of these paths will extend from the base point. Thus cutting and straightening these paths yeilds a polygon with $2g$ sides [33]. (In the event a multiple saddle occurs, the saddle is counted with multiplicity and a gradient descent path is extended from each connected component of its lower link.)

4.5.2 Constructing a Base Domain

The base domain of a polygonal mesh is a highly simplified representation. The base domain is often constructed as the result of a long sequence of mesh simplification steps, and sits at the root of a surface's multiresolution hierarchy. The faces of a base domain correspond to clusters in the mesh, and mapping each face into the plane leads to a multiple chart texture atlas. A *combinatorial* base domain is an abstract 2D cell complex X and is often also a simplicial complex. The MAPS method for surface parametrization [58] forms a combinatorial simplicial base domain as the end result of repeated vertex removals [25]. Though the base domain constructed by MAPS simplification matches the topology of the refined mesh, the organization of the domain is rather arbitrary. The Morse complex embedded in the mesh can be used to construct a combinatorial base domain. The faces of the complex may not be triangular, but the Morse-Smale complex [29] provides a method for their tessellation. We can trace gradient as-

cent paths extending from the uphill sides of each saddle, and these uphill paths will lead to a maxima. For a saddle, we construct an ordered list of alternating minima and maxima that are the endpoints of paths extending from the alternating uphill and downhill paths in counterclockwise order about the saddle. We tessellate the 2-cells of the complex with these saddle-minimum-maximum triangles[3].

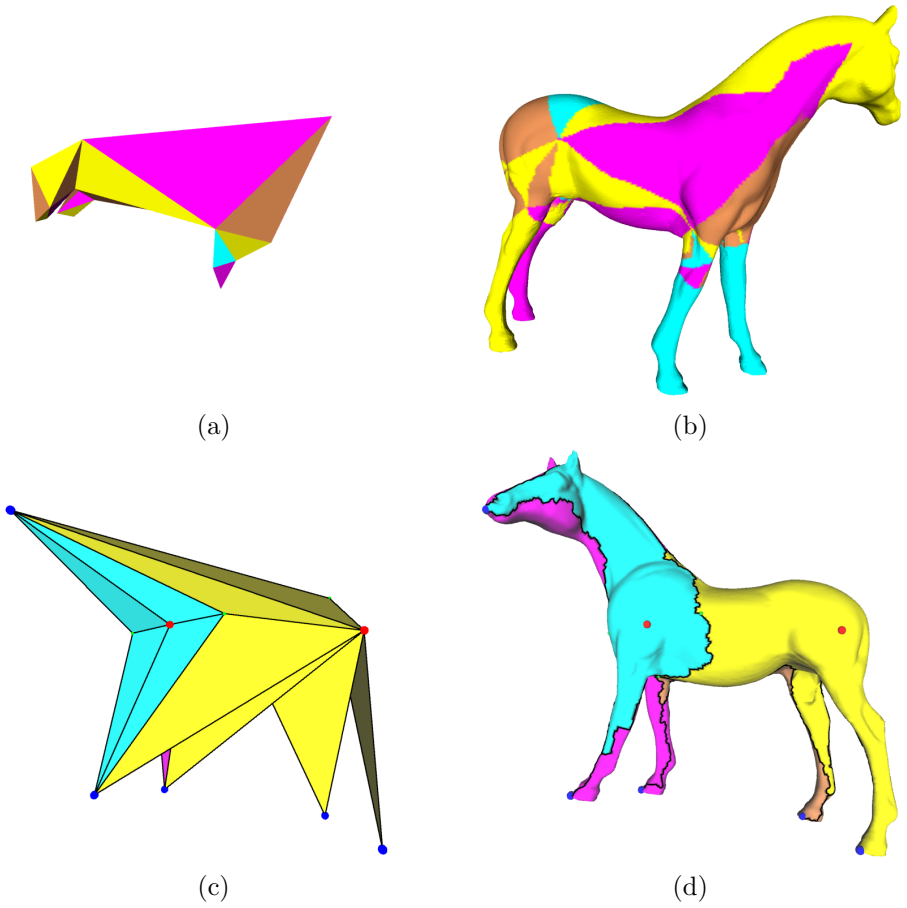

Figure 4.11: The greedy base domain (a) corresponding to face clusters (b) constructed by MAPS. Morse fairing allows the user to specify base domain vertices (minima) at feature tips (e.g. nose, feet) and faces (maxima) at feature areas (e.g. shoulder, hip) to create a more geometrically representative base domain (c) corresponding to face clusters (d).

The two steepest descent flows extending from the downhill sides of a saddle can lead to the same minimum, and likewise the two steepest ascent flows can lead to the same maximum, though both conditions cannot occur simultaneously for an individual Morse saddle. The geometry of the triangulated base domain may self-intersect and may contain degenerate or inverted triangles, but nonetheless serves as a viable combinatorial topological base domain for the

[3]A Morse saddle is surrounded by four such triangles, creating the "quad" structure used for multires topology processing [15].

purposes of supporting a parametrization. Moreover, as demonstrated in Figure 4.11, Morse fairing allows the user to pick extrema which can preserve features in the base domain lost by otherwise local and greedy simplification steps. The quality of some multiresolution algorithms depend directly on the quality of the base domain. For example, multiresolution mesh morphing [57] relies on the ability to construct meaningful correspondences between the base domains of a source and target object. Using Morse fairing to preserve features in a base domain makes it easier to find good source-target correspondences.

4.5.3 Clustering

When constructing an atlas, it is often desirable to form charts that approximate developable patches. Multichart Geometry Images [85] are generated from a rather expensive curvature clustering algorithm that formed clusters using Dijkstra's algorithm, then repeatedly recenters these clusters using Lloyd's algorithm.

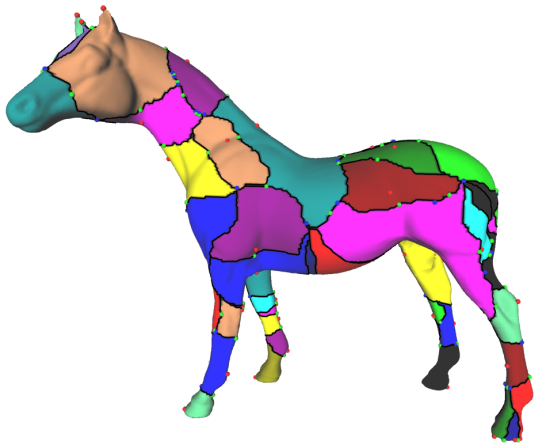

Figure 4.12: Morse complex of Laplacian-smoothed squared Gaussian curvature yields a rapid clustering toward developable charts.

The Morse complex of Laplacian-smoothed negated squared Gaussian curvature provides a more rapid clustering toward developable patches. Maximal regions will occur at developable regions whereas minima mark vertices of maximal curvature. Sheffer & Hart [90] showed that forcing chart boundaries through high curvature vertices helps minimize atlas distortion. The paths of the Morse complex will likewise pass through the minima found at high curvature regions. An iterated pegged Laplacian simplifies the Morse complex to the more persistent critical vertices corresponding to curvature features in the mesh, as demonstrated in Figure 4.12.

4.5.4 Visualization

We conclude the applications with the fair Morse complex of the cooling tunnels on a jet engine turbine shown in Figure 4.13. This polygonal mesh was reconstructed from volume data and contained 294 extraneous small simply-connected components most of which were tetrahedra that we were able to identify and remove. What remained was a single genus-165 component. The 330 paths of the fair Morse complex work their way around or through each of the 165 tunnels in the dataset and could be used for example as flythrough paths for visual inspection.

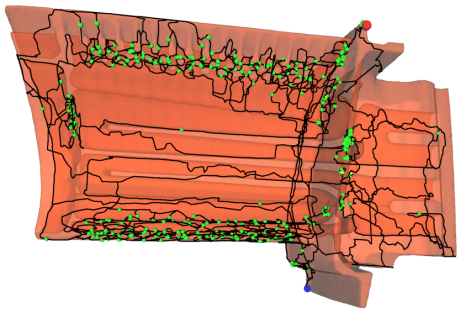

Figure 4.13: The Morse complex of a genus-165 turbine.

4.6 Conclusion

Morse theory is an exciting and powerful mathematical tool for reasoning about the global topological structure of a shape based only on local differential information. The unbounded number of critical points has thus far limited the application of Morse theory to a wider variety of problems in computer graphics, mesh processing and scientific computing. We have overcome this limitation with the concept of a fair Morse function that minimizes its variation across a manifold to produce the least possible number of critical points. Though we have made Morse theory on meshes more robust by now handling simply-connected flat regions, preventing gradient descent paths from reaching saddles and temporarily patching boundaries, we still find it necessary to repair meshes for the method to work properly. Though the critical points and paths can be made to handle non-manifold cases, these special cases make it more difficult to reason about the mesh geometry based on its Euler characteristic, and often our implementation contains hard-coded dependencies based on this reasoning. Stratified Morse theory provides insight into the application of Morse theory to cell complexes and perhaps the key to extending these techniques beyond manifolds.

4.6.1 Future Work

The 1-cells of the Morse complex are simple gradient descent paths and are by no means optimal. We believe that the gradient descent paths of a harmonic Morse function are probably geodesic, but leave the precise formulation, analysis and proof of such a statement for a future manuscript. Shortening the 1-cell loops like rubberbands would lead to shorter cuts, but these cuts may still not be the most basic cuts, as they can loop around any number of torus holes any number of times. Computational homology provides a way to optimize these cuts to avoid multiple loops and multiple holes [18], but we leave its integration into Morse fairing for future study. It is natural to consider the extension of these techniques to piecewise linear 3-manifolds and tetrahedral meshes. For example, Kartasheva [49] investigated cutting a tetrahedral-mesh solid into a topological 3-ball through the application of homology, and Edelsbrunner *et al.* [27] applied their persistence-based topology simplification to 3-manifolds. Unfortunately, 3-manifolds contain two kinds of saddles and its Euler characteristic is the difference between the minima plus one kind of saddle, and the maxima plus the other kind of saddle. Thus one can have a restricted number of extrema and an unbounded number of saddles and still satisfy the Euler charateristic. (If this were not the case, then Morse fairing would have led to a very easy algorithm for the classification of 3-manifolds.)

References

[1] D. Adalsteinsson and J. Sethian. A fast level set method for propagating interfaces. *J. Comput. Phys.*, 118:269–277, 1995.

[2] Burak Aksoylu, Andrei Khodakovsky, and Peter Schroeder. Multilevel solvers for unstructured surface meshes. *Siam J. Sci. Comput.*, (in review), 2003.

[3] Eugenio Aulisa, Sandro Manservisi, and Ruben Scardovelli. A mixed markers and volume-of-fluid method for the reconstruction and advection of interfaces in two-phase and free-boundary flows. *J. Comput. Phys.*, 188:611–639, 2003.

[4] Ulrike Axen and Herbert Edelsbrunner. Auditory morse analysis of triangulated manifolds. In Hans-Christian Hege and Konrad Polthier, editors, *Mathematical Visualization*, pages 223–236. Springer-Verlag, Heidelberg, 1998.

[5] J.A. Baerentzen and N.J. Christensen. Volume sculpting using the level-set method. In *Proc. Shape Modeling Intl. 2002*, pages 175–182, May 2002.

[6] J.A. Baerentzen and N.J. Christensen. Volume sculpting using the level-set method. In *Proc. Shape Modeling Intl. 2002*, pages 175–182, May 2002.

[7] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel Schikore. Visualization of scalar topology for structural enhancement. *IEEE Visualization '98*, pages 51–58, 1998.

[8] Chandrajit L. Bajaj and Daniel R. Schikore. Topology preserving data simplification with error bounds. *Computers and Graphics*, 22(1):3–12, 1998.

[9] Thomas F. Banchoff. Critical points and curvature for embedded polyhedral surfaces. *American Mathematical Monthly*, 77:475–485, 1970.

[10] D. M. Bates and D. G. Watts. *Nonlinear Regression and Its Applications*. Willey, New York, NY, USA, 1988.

[11] James F. Blinn. A generalization of algebraic surface drawing. *ACM TOG*, 1(3):235–256, 1982.

[12] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM Trans. on Graphics*, 22(3):912–924, July 2003. (Proc. SIGGRAPH 2003).

[13] David E. Breen and R. T. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE TVCG*, 7(2):173–192, April 2001.

[14] David E. Breen and R. T. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE TVCG*, 7(2):173–192, Apr. 2001.

[15] P.-T. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci. A multiresolution data structure for two-dimensional Morse-Smale functions. *Proc. Visualization 03*, pages 139–146, 2003.

[16] Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[17] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *Proceedings of IEEE Visualization*, pages 397–405, 2000.

[18] Eric Colin de Verdière and Francis Lazarus. Optimal system of loops on an orientable surface. *Proc. Foundations of CS*, pages 627–636, 2002.

[19] Mathieu Desbrun, Mark Meyer, and Pierre Alliez. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum*, 21:209–218, Sep. 2002. (Proc. Eurographics 2002).

[20] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, 1999.

[21] Mathieu Desbrun, Mark Meyer, Peter Schroeder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. *Proc. SIGGRAPH 99*, pages 317–324, 1999.

[22] Tamal K. Dey, Herbert Edelsbrunner, and Sumanta Guha. Computational topology. In B. Chazelle, J.E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*. Providence, 1999.

[23] Tamal K. Dey, Herbert Edelsbrunner, and Sumanta Guha. Computational topology. In B. Chazelle, J.E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, Providence, 1999. Contemporary Mathematics, AMS.

[24] Tamal K. Dey and H. Schipper. A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection. *Discrete and Computational Geometry*, 14:93–110, 1995.

[25] D. Dobkin and D. Kirkpatrick. A linear algorithm for determining the separation of convex polyghedra. *J. of Algorithms*, 6:381–392, 1985.

[26] Haixia Du and Hong Qin. Interactive shape design using volumetric implicit PDEs. In *Proc. Solid Modeling 2003*, pages 235–246, 2003.

[27] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale complexes for piecewise linear 3-manifolds. *Proc. Symp. on Computational Geometry*, pages 361–370, 2003.

[28] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete and Computational Geometry*, 30(1):87–107, 2003.

[29] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28(4):511–533, 2002.

[30] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comp. Phys.*, 183:83–116, 2002.

[31] Douglas Enright, Steve Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM TOG*, 21(3):721–728, 2002. (Proc. SIGGRAPH 2002).

[32] Jeff Erickson and Sariel Har-Peled. Optimally cutting a surface into a disk. *Proc. ACM Symp. on Comp. Geom.*, pages 244–253, 2002.

[33] P.A. Firby and C.F. Gardiner. *Surface Topology.* Ellis Horwood, New York, 1991.

[34] Kurt W. Fleischer, David H. Laidlaw, Bena L. Currin, and Alan H. Barr. Cellular texture generation. In *Proc. SIGGRAPH 95*, pages 239–248, 1995.

[35] Michael S. Floater. Mean value coordinates. *Computer-Aided Geometric Design*, 20(1):19–27, March 2003.

[36] M.S. Floater. Parameterization and smooth approximation of surface triangulations. *Computer-Aided Geometric Design*, 14(4):231–250, 1997.

[37] Nick Foster and Ronald Fedkiw. Practical animation of liquids. *Proc. SIGGRAPH 2001*, pages 15–22, Aug. 2001.

[38] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. SIGGRAPH*, pages 249–254, July 2000.

[39] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH 97*, pages 209–216, Aug. 1997.

[40] J. Glimm, J. Grove, X. L. Li, and D. C. Tan. Robust computational algorithms for dynamic interface tracking in three dimensions. *SIAM J. Sci. Comp.*, 21:2240–2256, 2000.

[41] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. *ACM Trans. on Graphics (Proc. SIGGRAPH 2002)*, 21(3):355–361, July 2002.

[42] Igor Guskov and Zoe Wood. Topological noise removal. *Proc. Graphics Interface*, pages 19–26, 2001.

[43] W. Hackbusch. *Multi-Grid Methods and Applications.* Springer-Verlag, 1985.

[44] J.C. Hart, E. Bachta, W. Jarosz, and T. Fleury. Using particles to sample and control more complex implicit surfaces. In *Proc. Shape Modeling International*, pages 129–136, 2002.

[45] Michael T. Heath. *Scientific Computing: An Introductory Survey.* McGraw-Hill, New York, 2nd edition, 2002.

[46] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. *Proc. SIGGRAPH 2001*, pages 203–212, 2001.

[47] Xiangmin Jiao. *Data Transfer and Surface Propagation in Multicomponent Simulations*. PhD thesis, University of Illinois at Urbana-Champaign, 2001. Tech Report UIUCDCS-R-2001-2251.

[48] Thouis R. Jones, Frédo Durand, and Mathieu Desbrun. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics (TOG)*, pages 943–949, 2003.

[49] Elena Kartasheva. The algorithm for automatic cutting of three-dimensional polyhedrons of h-genus. *Proc. Shape Modeling Intl. 99*, pages 26–33, 1999.

[50] R. Kimmel and A. Bruckstein. Shape offsets via level sets. *CAD*, 25(3):154–161, 1993.

[51] R. Kimmel, D. Shaked, N. Kiryati, and A.M. Bruckstein. Skeletonization via distance maps and level sets. *Comp. Vision and Image Understanding*, 62(4):382–391, 1995.

[52] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. *Proc. SIGGRAPH 98*, pages 105–114, 1998.

[53] Rvachev V L. Geometric applications of logic algebra. *Naukova Dumka*, 1967.

[54] J. Lawrence and T. Funkhouser. A painting interface for interactive surface deformation. In *Proceedings of Pacific Graphics*, pages 141–150, 2003.

[55] Jason Lawrence and Thomas Funkhouser. A painting interface for interactive surface deformation. In *Proc. Pac. Graphics 2003*, pages 141–150, Oct. 2003.

[56] Francis Lazarus, Michel Pocchiola, Gert Vegter, and Anne Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. *ACM Symp. on Comp. Geom.*, pages 80–89, 2001.

[57] Aaron W. F. Lee, David Dobkin, Wim Sweldens, and Peter Schröder. Multiresolution mesh morphing. *Proc. SIGGRAPH 99*, pages 343–350, 1999.

[58] Aaron W. F. Lee, Wim Sweldens, Peter Schroeder, Lawrence Cowsar, and David Dobkin. MAPS: multiresolution adaptive parameterization of surfaces. *Proc. SIGGRAPH 98*, pages 95–104, 1998.

[59] Aaron E. Lefohn, Joe M. Kniss, Charles D. Hansen, and Ross T. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proc. Vis. 2003*, pages 75–82, Oct. 2003.

[60] Shengtai Li and Linda Petzold. Moving mesh methods with upwinding schemes for time-dependent PDEs. *J. Comput. Phys.*, 131:368–377, 1996.

[61] Takashi Maekawa. An overview of offset curves and surfaces. *Computer-Aided Design*, 31:165–173, 1999.

[62] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John Hughes. Skin: A constructive approach to modeling free-form shapes. In *Proc. SIGGRAPH 99*, July 1999.

[63] Miriah D. Meyer, Pierre Georgel, and Ross T. Whitaker. Robust particle systems for curvature dependent sampling of implicit surfaces. In *SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005 (SMI' 05)*, pages 124–133, Washington, DC, USA, 2005. IEEE Computer Society.

[64] John W. Milnor. *Morse Theory*. Princeton Univ. Press, Princeton, NJ, 1963.

[65] Konstantin Mischaikow. Conley index theory. In *Dynamical systems (Montecatini Terme, 1994)*, volume 1609 of *Lecture Notes in Math.*, pages 119–207. Springer, 1995.

[66] Konstantin Mischaikow and Marian Mrozek. Conley index. In *Handbook of dynamical systems, Vol. 2*, pages 393–460. North-Holland, 2002.

[67] Bryan Morse, Terry Yoo, Penny Rheingans, D. Chen, and K.R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proc. Shape Modeling International*, pages 89–98, 2001.

[68] Shigeru Muraki. Volumetric shape description of range data using blobby model. In *Proc. SIGGRAPH 91*, pages 227–235, 1991.

[69] K. Museth, D.E. Breen, R.T. Whitaker, and A.H. Barr. Level set surface editing operators. *ACM TOG*, 21(3):330–338, July 2002. (Proc. SIGGRAPH 2002).

[70] Duc Quang Nguyen, Ronald P. Fedkiw, and Henrik Wann Jensen. Physically based modeling and animation of fire. *ACM TOG*, 21(3):721–728, 2002. (Proc. SIGGRAPH 2002).

[71] Duc Quang Nguyen, Ronald P. Fedkiw, and Henrik Wann Jensen. Physically based modeling and animation of fire. *ACM TOG*, 21(3):721–728, 2002. (Proc. SIGGRAPH 2002).

[72] Hitoshi Nishimura, Makoto Hirai, Toshiyuki Kawai, Toru Kawata, Isao Shirakawa, and Koichi Omura. Object modeling by distribution function and a method of image generation. In Proc. of *Electronics Communication Conference '85*, pages 718–725, 1985. (Japanese).

[73] Y. Ohtake, A. G. Belyaev, and H.-P. Seidel. Mesh smoothing by adaptive and anisotropic Guassian filter. In *Vision, Modeling and Visualization*, pages 203–210, 2002.

[74] Stanley Osher and Ronald Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2003.

[75] D. L. Page, A. F. Koschan, Y. Sun, J. K. Paik, and M. A. Abidi. Robust crease detection and curvature estimation of piecewise smooth surfaces from triangle mesh approximations using normal voting. In *Proc. Intl. Conf. on Computer Vision and Pattern Recognition*, volume 1, pages 162–167, 2001.

[76] B. Pham. Offset curves and surfaces: A brief survey. *CAD*, 24(4):223–229, 1992.

[77] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[78] G. Piponi and D. Borshukov. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. *Proc. SIGGRAPH 2000*, pages 471–478, 2000.

[79] S. Popinet. Gerris: A tree-based adaptive solver for the incompressible Euler equations in complex geometries. *J. Comp. Physics*, 190:572–600, 2003.

[80] Vaughan Pratt. Direct least-squares fitting of algebraic surfaces. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 145–152, New York, NY, USA, 1987. ACM Press.

[81] E. G. Puckett, A. S. Almgren, J. B. Bell, D. L. Marcus, and W. J. Jider. A high-order projection method for tracking fluid interfaces in variable density incompressible flows. *J. Comput. Phys.*, 130:269–283, 1997.

[82] Nicolas Ray and Bruno Levy. Hierarchical least squares conformal map. *Proc. Pacific Graphics*, pages 263–270, 2003.

[83] W. J. Rider and D. B. Kothe. Roconstructing volume tracking. *J. Comput. Phys.*, 141:112–152, 1998.

[84] M. Rudman. Volume-tracking methods for interfacial flow calculations. *Int. J. Numer. Meth. Fluids*, 24:671–691, 1997.

[85] P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe. Multi-chart geometry images. *Proc. Symp. Geom. Proc.*, pages 146–155, 2003.

[86] Guillermo Sapiro. *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press, 2001.

[87] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, New York, 1999.

[88] Vadim Shapiro and Igor Tsukanov. Implicit functions with guaranteed differential properties. pages 258–269.

[89] Alla Sheffer and Eric De Sturler. Surface parameterization for meshing by triangulation flattening. *ACM Trans. on Graphics*, 21(4):874–890, 2002.

[90] Alla Sheffer and John C. Hart. Seamster: inconspicuous low-distortion texture seam layout. *Proc. Visualization '02*, pages 291–298, 2002.

[91] Yoshihisa Shinagawa, Tosiyasu L. Kunii, and Yannick L. Kergosien. Surface coding based on morse theory. *IEEE Computer Graphics & Applications*, 11(5):66–78, 1991.

[92] Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. *Proc. SIGGRAPH 97*, pages 279–286, 1997.

[93] D. Steiner and A. Fischer. Topology recognition of 3d closed freeform objects based on topological graphs. *Proc. Pacific Graphics*, pages 82–88, October 2001.

[94] John Strain. Fast tree-based redistancing for level set computations. *J. Comput. Phys.*, 152:648–666, 1999.

[95] John Strain. Tree methods for moving interfaces. *J. Comput. Phys.*, 151:616–648, 1999.

[96] John Strain. A fast modular semi-Lagrangian method for moving interfaces. *J. Comput. Phys.*, 161:512–528, 2000.

[97] John Strain. A fast semi-Lagrangian contouring method for moving interfaces. *J. Comput. Phys.*, 169:1–22, 2001.

[98] T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher. Geometric surface processing via normal maps. *ACM Transactions on Graphics*, pages 125–132, 2003.

[99] T. Tasdizen, R. Whitaker, P. Burchard, and Stanley Osher. Geometric surface smoothing via anisotropic diffusion of normals. *Proc. Vis. 2002*, Oct. 2002.

[100] G. Taubin. Linear anisotropic mesh filtering. Technical report, IBM Research Division, Yorktown Heights, NY, 2001. Tech. rep. RC22213.

[101] Gabriel Taubin. A signal processing approach to fair surface design. *Proc. SIGGRAPH 95*, pages 351–358, 1995.

[102] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan. A front-tracking method for the computations of multiphase flow. *J. Comput. Phys.*, 169:708–759, 2001.

[103] G. Vegter. Computational topology. In *Handbook of Discrete and Computational Geometry*, pages 517–536. CRC Press, 1997.

[104] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proc. SIGGRAPH 94*, pages 269–278, July 1994.

[105] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *Visual Computer*, 2(4):227–234, 1986.

[106] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Feature-based surface parameterization and texture mapping. Technical Report GVU 03-29, Georgia Tech., 2003.

[107] Hong-Kai Zhao, Stanley Osher, and Ronald Fedkiw. Fast surface reconstruction using the level set method. *Proc. Workshop on Variational and Level Set Methods in Computer Vision*, July 2001.

Author's Biography

Xinlai Ni was born on August 10th, 1978 in Shenyang, Liaoning province, China. In 1997, he was selected as one of the five representatives on behalf of China to attend the 28th International Physics Olympiad in Sudbury, Canada, and won the silver medal award. Mr. Ni was exempted from the national college entrance examination due to this award and admitted to the Department of Computer Science and Technology at Tsinghua University, Beijing. He graduated with honor and obtained the degree of *B.S. in Computer Science* in 2001 from Tsinghua University, after which he joined the Ph.D program in the Computer Science Department at the University of Illinois at Urbana Champaign. His research topic was Computer graphics, in particular, 3D surface modeling and computational topology. His publication appeared in ACM Siggraph, International Meshing Roundtable, etc. Besides his role as a research assistant, Mr. Ni had also been serving as teaching assistant in various CS courses such as Data Structure and Computer Graphics. He had summer internships at Microsoft in the year 2005 and 2006. In May 2007, Mr. Ni will be awarded both the Ph.D degree in Computer Science and the Master of Science degree in Mathematics.